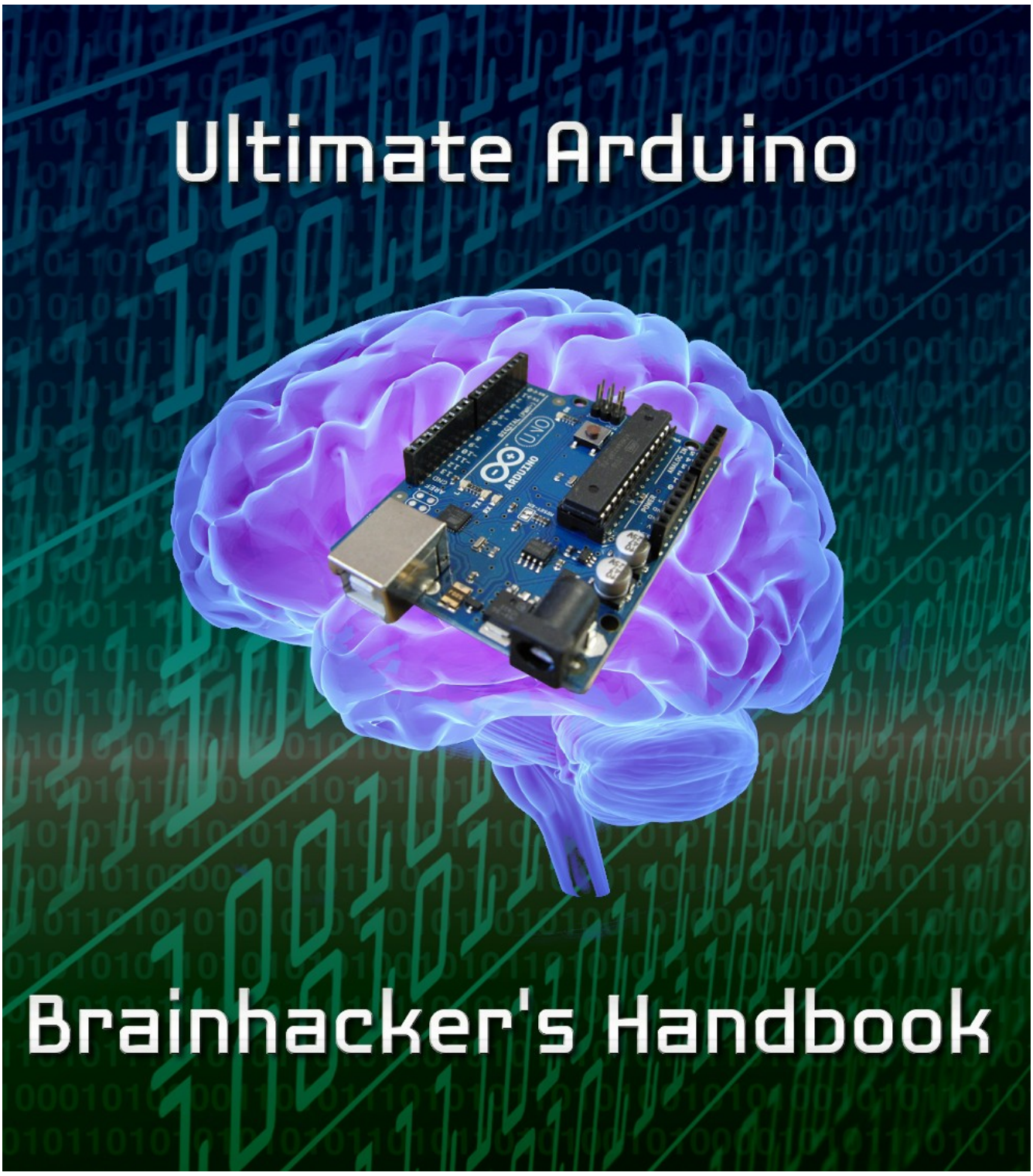
Ultimate Arduino



Brainhacker's Handbook

v1.0.0

Contents

Published By Mark Maffei

Table Of Contents

Legal Notices & Warnings (Please **read carefully** before proceeding)

Acknowledgements & Attribution

Part 1: Brainwave Entrainment Basics

- **Brief Introduction**
- **Brainwave Entrainment Simplified**
- **Binaural Beats vs. Isochronic Tones**
- **Scientific Benefits Of Brainwave Entrainment Technology**

Part 2: Single Individual Frequency Effects

- **Reputed Effects Of Single Individual Brainwave Frequencies**
- **The Brain's "Sweet Spot" Operating System Frequency**

Part 3: Cossum's Awesome Enhanced Brain Machine

Part 4: The Art Of EEG Toy Hacking Gone Wild!

- **How To Hack Toy EEGs**
- **Visualizing Mind Activity With A Hacked Toy EEG**
- **Full RAW EEG Data From The MindFlex Headset Hack**
- **Hacking MindWave Mobile**

Part 5: Connecting Arduino to Processing

Legal Notices & Warnings

Legal Notices:

You are permitted to remix this report provided that:

- You give *prominent credit and link love* where credit's due, and prominently pass these same CC BY SA rights onto others within your remix;

- You use a *completely different title and ecover design*, and **do not** in any way insinuate that Mark Maffei endorses your remix in any manner whatsoever.

The Publisher of this report has striven to be as accurate and complete as possible in its creation. However, *this does not warrant or represent at any time that the contents within are suitable for all purposes and intents.*

As a reader of this report, you understand and agree to the following; if not, *simply discontinue using this report*:

- The Publisher and individual authors of this report *will not be responsible for any losses or damages of any kind incurred by the reader*; whether directly or indirectly arising from the use of the information found in this report. Anything you do with the following information is **at your own personal risk**.

- The Publisher of this report reserves the right to make changes *without notice at his sole discretion and assumes no responsibility or liability whatsoever* on the behalf of the reader of this report.

- Readers are advised to *exercise due diligence and seek professional help when prudent to do so.*

Warnings:

- Those under 26 years of age (and most especially children) are advised *extreme caution* with brainwave entrainment, as the brain is still developing and is more sensitive.

- If you are prone to seizures/epilepsy or have an irregular heartbeat (especially if you have a pacemaker), you are advised *against* brainwave entrainment.

- Pregnant women should *consult their physician* when considering using brainwave entrainment.

Simply Stated: *This information is for educational, entertainment, and informational research purposes only.*

Acknowledgements & Attribution

First, a salute to the following brilliant brainhackers:

- Lady Ada
- Eric Mika
- Sophi Kravitz
- Mitch Altman
- Bobby Cossum
- Darren Mothersele

A hearty “Thank you!” especially goes out to Adafruit, Arduino.cc, Makezine.com, Hackster.io and SparkFun; all of which are paramount in the *ongoing research and development* of the Ultimate Arduino Brainhacker’s Handbook.

Last, but certainly not least, this tasty tome of brainhacker brilliance is warmly dedicated to Arduino brainhackers of all persuasions both great and small: May your brainhacking adventures always be pleasurably rewarding!

Attribution

The Ultimate Arduino Brainhacker’s Handbook was compiled by Mark Maffei in 2017 via the Creative Commons BY SA deed:

Please redistribute generously!

Pt. 1: Brainwave Entrainment Basics

Attribution: Written by Mark Maffei via CC BY SA; images belong to their respective owners.

Brief Introduction

Everything you experience is both affected by and affects your brainwaves; hence a distinct relationship between your brain frequencies and your perceptual life experiences dynamically shifts moment by moment.

Said another way:

- Anything that radically changes your perception radically changes your brainwave rhythms; whether it be psycho-meds, recreational drugs, sex, meditation, solving puzzles, etc.

- Conversely, anything that radically alters your brainwaves, radically alters your mood and perception.

Therefore, understanding *optimal brainwave frequency profiles* is the first step to unleash the true power of your mind to do amazing Ψ stuff, rapidly heal, consciously manifest your desires, etc.

For example, your success with relaxing, focusing, and getting into the Zone will expedite your subconscious mind accepting your visual and verbal suggestions; whereby effectively changing its programming (i.e. removing all desire for burdensome additions, speeding up the healing process, instilling desirable habits, speed-learning new skills, etc.)

With a combination of the right tools, knowledge, and some practice... you too can quickly get your brain waves into the Zone **at will**. Then it's simply a matter of implanting the new thoughts and behaviors you want via congruent visual and auditory suggestions.

As luck would have it... you've definitely come to the right place; so without any further ado, let's get to it!

Brainwave Entrainment Simplified

So what exactly is “brain entrainment”?!

From BrainWorksNeurotherapy.com:

"Brainwave entrainment is a method to stimulate the brain into entering a specific state by using a pulsing sound, light, or electromagnetic field. The pulses elicit the brain’s ‘frequency following’ response, encouraging the brainwaves to align to the frequency of a given beat."

Simply put, brainwave entrainment (aka "**hemi-sync**") is a type of resonance ; whereby two unsynchronized brainwave rhythms will synchronize with each other in a relatively short amount of time.

You're likely familiar with the proverbial hypnotist mesmerizing their subject with a pocket watch swinging back and forth like a pendulum. Interestingly, it actually has historical roots.

Back in the 1656, Christian Huygens discovered that two pendulums started at different times (completely out of sync) would synchronize, swinging in opposite directions; hence he ended up stumbling upon the realization that brainwave frequencies can also resonate with an external stimulus.

Here's a really short, yet strangely satisfying video demonstrating this pendulum effect.

On a related side note, the main reason certain individuals have gotten less than favorable results (as compared to what they were hoping for) is because *they approach brain entrainment all wrong*.

This is why it is important to utilize the right type entrainment for the job:

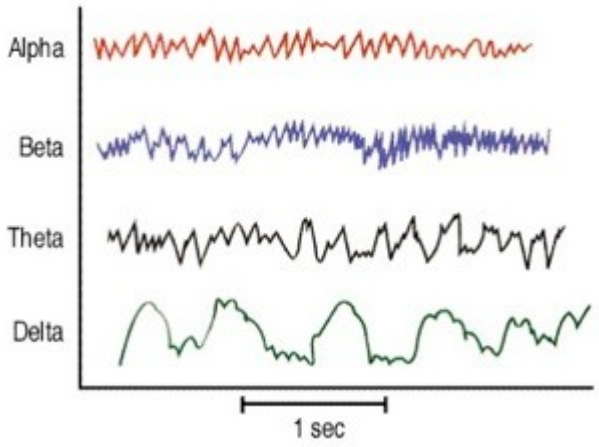


Image via BinauralBrains.com

- **Beta stimulation**: Beta wave stimulation can improve attention, relieve short-term stress, decrease headaches, and decrease behavioral problems and emotion-based exhaustion, amongst other benefits.

- **Alpha stimulation**: Alpha wave stimulation benefits include relaxation, stress reduction, pain relief, and improved recognition, to name just a few.

- **Theta stimulation**: Theta wave stimulation is also great for deep relaxation, as well as exploring lucid dreaming, altered states of consciousness, and Ψ experimentation, for example.

- **Delta stimulation**: Delta wave stimulation can improve short-term stress, as well as relief from headaches/migraines, etc.

Binaural Beats vs. Isochronic Tones

Both binaural beats and isochronic tones can initiate hemi-sync; and neither is necessarily 'better' or 'worse' than the other. Specific case-use and how each gets the job done does varies greatly, however.

Binaural Beats: Binaural beats work by broadcasting a different frequency into each ear.

The difference between the two produces a third **beat frequency**. For example, playing a 124Hz frequency in one ear and 114Hz in the other produces a 10Hz internal beat frequency:

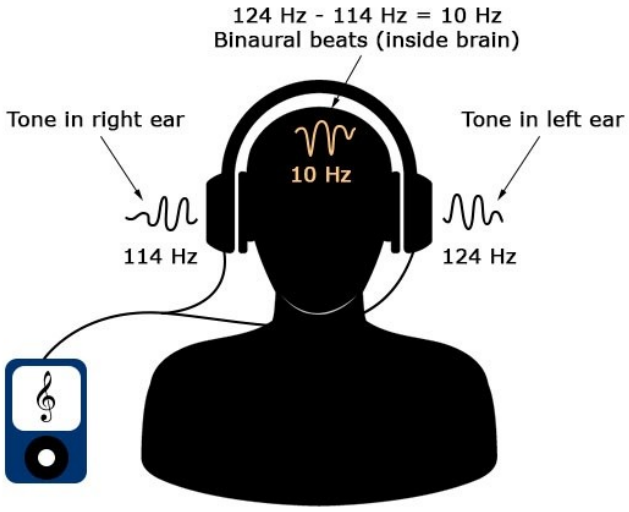


Image via BeBrainFit.com

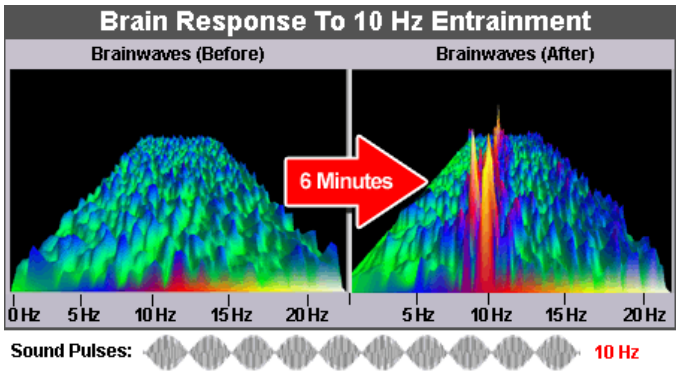


Image via TransparentcCorp.com

Math Geek Alert: https://en.wikipedia.org/wiki/Beat_(acoustics)#Binaural_beats

Unlike isochronic tones, binaural beats have a *continuous wave pattern* and therefore tend to blend better with ambient music (making them the popular choice for hypnosis CD recordings and the like).

When listening to binaural beats, stereo headphones are **highly desirable** for maximum benefit in the shortest amount of time. However, centering yourself between *two equi-distantly spaced stereo speakers* (relative to your ears) will also work; although it can take awhile longer to get the equivalent results, as compared to if you were wearing head-phones.

Likewise, closing your eyes is not absolutely necessary, albeit it also greatly helps you to gain maximum benefit in the shortest amount of time (considerably less distractions for the brain to deal with).

Isochronic Tones: Isochronic tones are monaural beats comprised of *evenly spaced pulses* that combine with a carrier frequency to make a single tone, and are designed to evoke a **frequency-following response**; whereby able to harmonize with the brain efficiently.

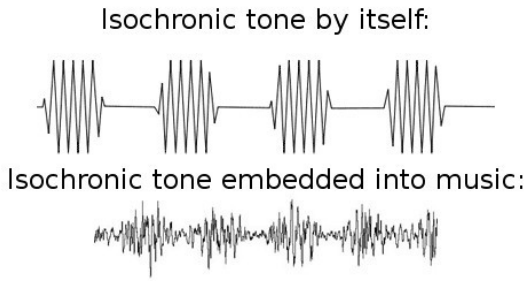


Image via MindAmend.com

Supposedly, these spaces make isochronic tones more efficient and effective than binaural beats, albeit they do initially take a bit of getting used to.

However, isochronic tones are *not dependent* upon headphones or spacing your speakers equidistantly from your ears to achieve the desired result. The important thing is that you are able to hear the tone.

In Short:

- Binaural beats are best for *submersive experimentation* (i.e. eyes closed, lights dimmed, and headphones on).
- Isochronic tones are definitely more convenient, and therefore ideally suited to set-and-forget sessions (i.e. eyes wide open, going about your business as usual).

And that's the *true beauty* of brainwave entrainment - it doesn't require any "effort" to use, once you've selected the right tone/pulse profile that suits your needs at the moment. Simply listen and let go with binaural beats, for a more "submersive" experience or go about your business as usual if listening to isochronic tones.

While still on the topic of isochronic tones, I highly encourage you to play around with the following two free online tools to gain a deeper understanding:

- https://mynoise.net/NoiseMachines/isochronicBrainwaveGenerator.php
- https://brainaural.com/

Also, here's a few more brainwave entrainment/relaxation toys to play with:

- Audio Check (Great little toolbox of goodies for calibrating your Arduino brainhacking projects, frequency measuring equipment, etc.)
- Chameleon (User-definable relaxing music generator)
- White Noise & Co. (User-definable white noise generator)
- Jungle Life (User-definable jungle noise generator)
- Rain Noise (User-definable rain noise generator)
- Tibetan Choir (User-definable overtone throat singing generator)

There's actually quite a few more here, but these are ones I wanted quick and easy access to. On a related side note, **light-based brainwave entrainment** (i.e. pulsating "stereo" LEDs) is the light-based version of binaural beats.

Brainhacker Trivia: Binaural beats were first discovered by Heinrich Wilhelm Dove in 1839

Scientific Benefits Of Brainwave Entrainment Technology

Published scientific findings concerning brainwave entrainment largely suggest that it may be a successful adjunct treatment when used with the *proper frequency profile*.

Quite frankly, this area of study is still very much in its infancy; nonetheless, here's a few interesting scientific insights I've come across:

- https://www.transparentcorp.com/downloads/Huang-PsychologicalEffectsBrainwaveEntrainment.pdf
- http://www.trainyourbraintransformyourlife.com/Alert%20clinical%20study.pdf

- http://www.researchgate.net/publication/247871689_Photic_Driving_and_Altered_States_of_Consciousness_An_Exploratory_Study
- http://www.ncbi.nlm.nih.gov/pubmed/6945531
- http://www.ncbi.nlm.nih.gov/pubmed/2708044
- http://onlinelibrary.wiley.com/doi/10.1111/j.1526-4610.1985.hed2508444.x/abstract
- http://www.scirp.org/journal/PaperInformation.aspx?paperID=36722#.VRsZxPnF9Dw
- http://www.ncbi.nlm.nih.gov/pubmed/16706812
- http://www.ncbi.nlm.nih.gov/pubmed/6945531
- http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0027030/
- http://www.ncbi.nlm.nih.gov/pubmed/18780583
- http://www.ncbi.nlm.nih.gov/pubmed/23862643

Brainhacker Trivia: In 1997, a Japanese television station broadcast a light and sound strobe-like effect embedded into a cartoon. Though lasting only a mere 5 seconds, 700 children ended up in hospital with seizures, and untold thousands more experienced headaches, dizziness, and nausea.

"With brain wave entrainment technology, changing brain wave states is an instantaneous and effortless process. The 'periodic stimulus' can be sound, vibrations and/or light.

We have found that we get the best results with blinking lights which are experienced through closed eyelids. This is only problematic for people with existing diagnosis of photo-induced epilepsy, as blinking lights can induce a seizure in them."

- David Mager

Editor Note: In regard to the above quote, light-based brain entrainment is definitely quicker (and often times more intense) than sound-based brain entrainment *on a one-to-one basis*.

However, if you want to *take it to the next level*, be sure to read the entire "**Cossum's Awesome Enhanced Brain Machine**" section. By properly combining both types of entrainment, truly powerful stuff happens.

Part 2: Single Individual Frequency Effects

Reputed Effects Of Single Individual Brainwave Frequencies

Attribution: The following list of frequencies in Hertz (Hz) is a massively stripped down and streamlined portion of the Brainwave/Cymatic Frequencies; originally compiled by Michael Triggs, and further enhanced by Mark Maffei via CC BY SA *provided this entire attribution remains fully intact*. However, you can slightly reword it to fit the flow of your remix.

- 0.5 - very relaxing against headache and lower back pain; Thyroid, reproductive, excretory stimulant, whole brain toner.
- 0.5-1.5 Pain relief; endorphins, better hypnosis
- 0.5-4 Trance, suspended animation, anti-aging*****. Reduces amount of cortisol, a hormone associated with stress & aging. Increases the levels of DHEA (anti-aging) & melantonin (decreases aging process).

In conjunction with other frequencies in a waking state: "Delta acts as a form of radar – seeking out information – reaching out to understand on the deepest unconscious level things that we can't understand through [conscious] thought process."

Conducive to miracle type healing, divine knowledge, inner being and personal growth, rebirth, trauma recovery; "one with the universe" experiences, near death experience; characterized by "unknowing", merely a blissful "being" state such as deep sleep or coma.

*****The anti-aging info comes from a Brainwave Generator preset authored by TheMind2 - he uses binaurals at 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 & 4.0 HZ. They all play simultaneously.

- 0.9 - Euphoria
- 0.95 - Whiplash
- 1.0 - Feeling of well-being, pituitary stimulation to release growth hormone; overall view of inter-relationships; harmony & balance.
- 1.05 - Helps hair grow + get its color back [RA]; pituitary stimulation to release growth hormone (helps develop muscle, recover from injuries, rejuvenation effects).
- 1.2 - Used on headaches.
- 1.45 - Tri-thalamic entrainment format. According to Ronald deStrulle, creates entrainment between hypothalamus, pituitary & pineal. May benefit dyslexics + people with Alzheimer's.

- 1.5 - Abrahams Universal Healing Rate; sleep; individuals whose ailments have manifested into the fourth stage of Chronic Fatigue, where some form of disease is apparent, experienced a release from the negative sensation of their symptoms when moved into 1.5Hz.
- 1.8 - Sinus Congestion seems to clear centering around 1.8 Hz (tested with binaural beats, primarily).
- 2.0 - Nerve regeneration.
- 2.06 - Associated with coccyx (small triangular bone at end of the spinal column).
- 2.30 - Associated with genitals.
- 2.5 - Pain relief, relaxation; production of endogenous opiates; use for sedative effect; reported use on bleeding, bruises, insomnia, and sinusitis. Sexual stimulation?
- 2.57 - Associated with bladder.
- 2.67 - Associated with intestines.
- 3.07 - Associated with the pelvis.
- 3.4 - Sound sleep.
- 3.44 - Associated with ovaries.
- 3.5 - Feeling of unity with everything; accelerated language retention; enhancement of receptivity; reduction in depression & anxiety; whole-being regeneration; DNA stimulation.
- 3.6 - Reduction in anger & irritability.
- 3.84 Associated with the liver/pancreas.
- 3.9 - Reduction in unsociable behavior; crystal clear meditation, lucid dreams, enhanced inner awareness, "facilitates easy access to inner resources & creates space for inner peace + self-renewal"; 1 octave below *traditional* Schumann Frequency (Earth resonance) @ ~7.8 Hz.
- 4.0 - Psi (**Ψ**) frequency “sweet spot"; seduction mindset; vital for memory & learning; subconscious problem solving/full memory scanning (if one can manage to stay awake)."Those who suffer from Chronic Fatigue exhaust very easily. When moved to 4HZ these individuals showed marked improvement in the length of time between the occurrence of exhaustion after certain exercises were completed."

FYI: According to Cavanagh (1972), theta at 4Hz corresponds to a *full memory search*. Theta, then, like alpha, is a scanning frequency. Cavanagh began by compiling a number of studies dealing with different classes of stimuli (digits, colors, letters, words, geometrical shapes, random forms, and nonsense symbols).

Each class of stimuli was found to have a characteristic reaction time. However, he found a constant of 243.2 ms when multiplying the reaction time for a single item by the maximum number of items in a given class. This indicated that each item class was scanned at a different speed, but that scanning of the full memory is always executed at a speed of 4Hz.

According to Webber (2001), after 10 minutes of listening to 4Hz theta waves, a very interesting “water effect” occurred in his brain once he stop listening, as well as increased sensitivity in his senses.

A suggestion might be found in the work of Patterson and Capel (1983) in Surrey, England.

They found that different neurotransmitters were triggered by different frequencies and wave forms. For example, a 10-hertz signal boosts production and turnover rate of serotonin.

"Each brain center generates impulses at a specific frequency, based on the predominant neurotransmitters it secretes," says Dr. Capel. "In other words, the brain's internal communications system--its language, if you like--is based on frequency..." (Ostrander & Schroeder, 1991).

The implications of Capel's & Patterson's work is that one can alter the brain's neuro-chemistry, and thereby it's functioning, with modifications of brain wave frequency.

- 4.11 - Associated with kidneys; strength.
- 4.125 – Associated with DNA restoration; harmonic of the Solfeggio tone **528Hz** (which is the exact frequency used by genetic biochemists to repair broken DNA) and recent Earth resonant frequency shift.
- 4.5 - Associated with Shamanic state of consciousness and Tibetan Buddhist chants.
- 4.6 - Associated with spleen & blood; emotional impulse.
- 4.9 -Associated with introspection; induce relaxation, meditation & deeper sleep.
- 5.0 - Associated with unusual problem solving; less sleep needed; theta sounds replace need for extensive dreaming; relaxed states; pain relief (beta endorphin increases of 10-50% reported); alleged sphincter resonance (??)
- 5.14 - Associated with stomach; emotional acceptance.

- 5.35 - Associated with lungs.
- 5.5 - Moves beyond knowledge to knowing; shows vision of growth needed ("inner voice" guidance; intuition.
- 5.8 – Reduces fear, absent-mindedness, dizziness.
- 6.0 - Associated with long term memory stimulation; reduces unwillingness to work.
- 6.15 - Associated with heart; love,warmth.
- 6.2 – 6.7: Frontal Midline Theta (Fm Theta) is a specific EEG frequency seen in those subjects actively engaged in cognitive activity, such as solving math problems & playing Tetris.
- 6.26 – 6.6: Hemispheric desync, confusion, anxiety, low Reaction Time, depression insomnia.
- 6.30 – Associated with telepathy mental/astral projection; accelerated learning; increased memory retention (??); reduce anger + Irritability (??)
- 6.8 - Associated with telepathy; possible use for muscle spasms.
- 6.88 Associated with collarbones; vitality; overall balance; stability.
- 7.0 – Associated with telepathy mental/astral projection; bending objects; psychic surgery; increased reaction time; mass aggregate frequency (i.e. can de-aggregate matter). Also alleged to resonate/rupture organs at excessive intensity; treatment of sleep disturbances; bone growth.
- 7.69 - Associated with shoulders; strength of the arms; expansion; teaching.
- 7.8 - Associated with ESP activation; Doyere's group (1993), found that short high frequency bursts at 7.7 Hz induced LTP in prefrontal cortex, though only for one day.
- 7.83 - Associated with anti-jetlag, anti-mind control, improved stress tolerance; psychic healing experiments; pituitary stimulation to release growth hormone (helps develop muscle, recover from injuries, rejuvenation effects); reports of accelerated healing/enhanced learning.
- 8.0 – Associated with DNA repair (RAD-6). Also 4Hz is a direct harmonic of 8Hz (covered above).
- 8.22 - Associated with mouth; speech; creativity.

- 8.3 - Pick up visual images of mental objects (??)
- 9.0 - Associated with awareness of body imbalance causes; dirct harmonic of 4.5Hz above.
- 9.19 - Associated with upper lip; emotions; conflict resolution.
- 9.4 - Major frequency used for prostate problems.
- 10 - Enhances release of serotonin; mood elevator; acts as analgesic; supposedly also good for hangovers & jet lag (and nicotine withdrawal, in one case). Decreased pain; arousal, increased alertness and sense of well being; significant improvements in memory, reading and spelling are reported (reportedly in conjunction with 18 HZ; 8Hz beat frequency). Also a direct harmonic of 5Hz covered above.
- 10.2 - Associated with catecholamines.
- 10.3 Associated with nasal passages; breathing; taste.
- 10.5 - Associated with healing of body; stimulating for the immunity.
- 10.6 - Relaxed & alert.
- 10.7 - Associated with ears; hearing; formal concepts.
- 11.0 - Used to achieve "relaxed yet alert" states; see also 5.5Hz entry above, as 11Hz is a direct harmonic.
- 12.3 - Associated with eyes; visualization.
- 13.0 – Alleged sphincter resonance (??).
- 13.8 - Associated with the seventh sense, final decision.
- 14.0 - Alert focusing; vitality; concentration on tasks; audio-visual stimulation alternating between 14 & 22 Hz?
- 15.0 – Allegedly helps alleviate chronic pain.
- 15.4 - Associated with cortex; intelligence.
- 16.0 - Bottom limit of normal hearing; release oxygen & calcium into cells; direct harmonic of both 4HZ & 8Hz (covered above).

- 20.0 – Stimulant; used on sinus disorders/infections, head cold, and headache; direct harmonic of both 5Hz & 10Hz (covered above).
- 20.215 - Allegedly can produce LSD 25-like effects and/or Ψ (??)
- 22.027 – Associated with increased insertion; possibly Ψ (??)
- 25.0 - Bypassing the eyes for image-imprinting (visual cortex); tested clinically with patients who complain of anxiety.
- 31.32 - Pituitary stimulation to release growth hormone (helps develop muscle, recover from injuries, rejuvenation effects).
- 32 - Desensitizer; enhanced vigor & alertness.
- 33 - Associated with Christ consciousness; hypersensitivity; harmonic of 16.5Hz, covered above.
- 38 – Endorphin release.

The Brain's "Sweet Spot" Operating System Frequency

Commonly referred to as the brain's "operating system", **40Hz** activity ranges from **38.8-40.1Hz**; regardless of the electrode location. The average frequency is in the ~39.5-Hz.

In summary, when the body is profoundly relaxed and the mind is in a state of high focus and concentration, 20Hz/40Hz brain activity can be seen in the raw and quantitative EEG of some subjects. Not surprisingly, 40Hz is associated with **being in the Zone**.

It is also associated with information-rich task processing and high-level information processing; dominant in fearful problem-solving situations; synchronization in rapidly processing incoming sensory stimulation, and also a harmonic of 20Hz, 10Hz and 5Hz .

"'For scientists who study the human brain, even its simplest act of perception is an event of astonishing intricacy. 40 Hz brain activity may be a kind of binding mechanism".

- Dr. Rodolfo Llinas, professor of neuroscience (New York University)

Llinas believes that the 40Hz wave serves to connect structures in the cortex (where advanced information processing occurs) and the thalamus, a lower brain region where complex relay and integrative functions are carried out.

In another study quite some time back called *40Hz brain activity, consciousness, and psi* in which gamblers were presented a computer screen with four playing cards shown on them, and then asked to guess which of the four cards would appear on the screen by itself next.

Not surprisingly, their guessing was about as accurate as random chance.

However, *when they had their eyes on the correct card*, the amount of 40Hz activity registering in their brain was significantly higher, and involved in the processing of **Ψ information**:

"EEG data recorded from 22 frequent gamblers while they played a laboratory, video-gambling game. On each hand, four cards were presented sequentially in the center of a video screen using an inter-stimulus interval of 2600 ms and a stimulus duration of 330 ms in counterbalanced wager and non-wager blocks.

Then, subjects guessed which card would later be selected as the target by a computerized, random process. Guessing accuracy was at chance level (,p = .465; one-tailed binomial). EEG analyses focused on 40Hz power in the 150-500 msec latency window following delivery of the target and non-target cards.

ANOVAs indicated greater 40Hz power for targets than for non-targets over left-frontal scalp in both wager and non-wager conditions, F (1/21) = 7.87, p = .005, and over the right-posterior scalp in the wager condition, F (1/21) = 5.81, p = .013, both one-tailed.

These findings indicate that:

1) 40Hz activity is involved in the processing of psi [ψ] information.

2) The attentional mechanisms of the brain utilized in focused arousal are also utilized in processing psi [ψ] information in the wager condition, although the left-frontal effect, found in both wager and non-wager conditions, is unlike the posterior-parietal locus of 40Hz effects observed by Sheer.

Neither the left frontal nor the right parietal loci of effects overlay primary sensory areas of the cerebral cortex, suggesting that the 40Hz effects we observed were not due to sensory processes.

The present EEG findings are consistent with the notion of unconscious or preconscious psi. Guessing accuracy was at chance level and yet 40Hz activity following target and non-target stimuli differed significantly.

Thus, although conscious behavior was not influenced by psi information, the differential brain responses indicated recognition at some level that the stimuli belonged to different categories. These findings in the frequency domain support and extend our previous findings with event-related brain potentials."

If the scientific aspect of the 40Hz “brain OS” fascinates you, this page cites a wealth of interesting studies and information.

40 Hz Gamma Pure Tone Binaural Beat: https://www.youtube.com/watch?v=ZGHbKWGgH_E

Pt. 3: Cossum's Awesome Enhanced Brain Machine

"I can see through mountains, watch me disappear, I can even touch the sky.
Swallowing colors of the sound I hear, am I just a crazy guy? You bet!"

- Ozzy Osbourne/"Flying High Again"

Image via Makezine.com

Attribution: The following adaptation by Mark Maffei is based on Bobby Cossum's documentation, of which both his documentation and code are licensed GPL; hence this adaptation is via CC BY SA, *provided this entire attribution remains fully intact*. However, you can slightly reword it, to fit the flow of your remix.

Brief SLM Introduction

Historically, microcontroller-based sound/light machines (SLM's) really became a "thing" a little over a decade ago (via **Mitch Altman's** infamous hack of Lady Ada's MiniPOV kit), called the "Brain Machine", which was featured in Make: 10. Here's a short but interesting interview of Altman: https://www.youtube.com/watch?v=vysfoePdiik

Based on the ATtiny2313P, It was an instant 'overnight sensation'; with thousands of Makers from around the world experiencing the pleasures of sound/light brain entrainment for the very first time. As to be expected, several custom mods based on his initial design sprouted up like magic mushrooms.

And while both projects (in their original form) have long since been depreciated, the original Brain Machine nonetheless deserves *historical recognition and honor* as the root of all modern Arduino-based SLM's.

However, of **equally great** historical relevance is the *amazingly awesome upgrades* **Bobby Cossum** unveiled a couple years later at the 2010 Maker Faire; directly based on Mitch Altman's original Brain Machine design.

Cossum's Awesome Enhanced Brain Machine Upgrades

Three of Cossum's upgrades that are especially noteworthy:

1. It was the first 100% open source **RGB-based SLM**;

2. It incorporates a brilliant method of turning the annoying-sounding previous square waves into *far more pleasant sounding* **pseudo-sine waves**;

3. Plus, his enhancements are compatible with both the original ATtiny-based version **and Arduino**.

Unfortunately, the code and documentation is spread out across multiple pages…

That is, until now! I have gone to *great lengths* to faithfully reconstruct Cossum's most excellent contribution to the Brainhacker Community below, in an easy-to-follow manner.

[**Editor Note:** Cossum's original documentation/notes will be in this font; editing as necessary for clarity/flow. Also, the **ATtiny-based** enhancements/mods directly correspond to Altman's original schematic below]

Cossum's SLM Brain Machine Documentation Remixed

An RGB Brain Machine! A screen saver for your head! Mitch Altman's Brain Machine from Make: 10 redesigned with sine wave-ish audio and RGB LEDs. This project can be built on a MiniPOV3 or on an Arduino. To build the code for a MiniPOV3, run the makefile in the archive. To build for an Arduino, unzip the archive into an Arduino project directory and rename the file slm.c to slm.pde.

[Download The Code]

Remixed From The Associated Wiki Page:

Introduction

This page describes an upgrade to the Brain Machine project featured in Make: 10. By applying the hardware and firmware modifications described, you will end up with a device with superior audio and firmware controllable LED brightness.

The updated firmware provides more pleasing sine wave-like audio using a technique known as direct digital synthesis. With the supplied firmware, beat frequencies from 14Hz down to 1Hz are available in 1Hz increments; with a base frequency of 200Hz. However, the code is easily modifiable; *no longer requiring hairy timer math* as with the original firmware.

Details

Tools Required:

- Soldering iron
- Desoldering braid
- Solder

Hardware Modification:

The hardware modification involves the removal of R4, R6, LED5 and the capacitor nearer th DB-9 connector. R6 and the capacitor will be re-used so try to remove them without damaging them. R4 and R6 are both labeled on the circuit board. LED5 is next to the capacitor that is not being removed.

Further:

Further and more interesting modifications to your Brain Machine are possible. The addition of RGB LEDs adds a whole new dimension to the experience. Unfortunately blue and green LEDs require a voltage higher than the 3 volts available from two AA alkaline batteries or the 2.4 volts from a pair of nickel metal hydride cells.

Remixed From The “README” File Contained Within slm.zip:

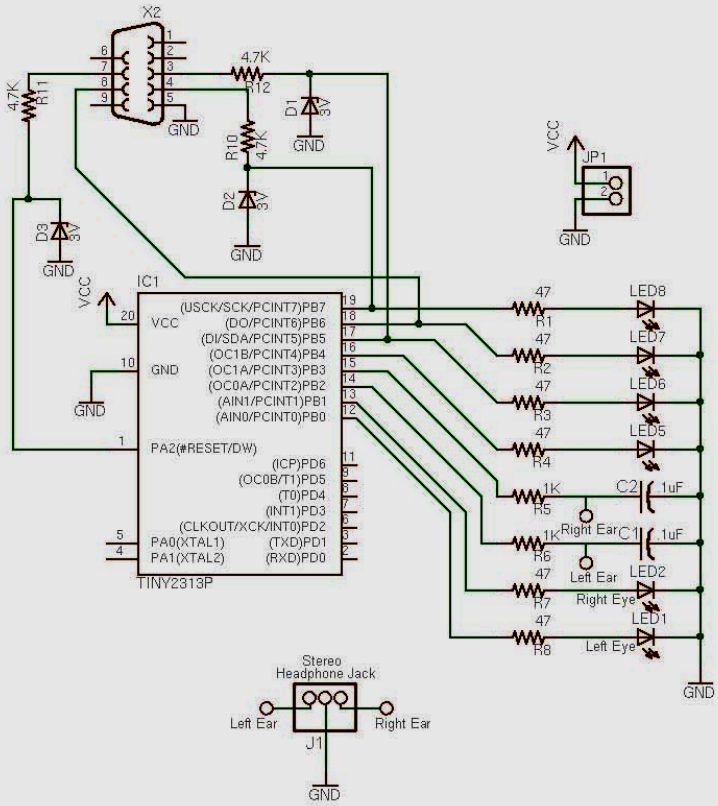
This file describes the process for rebuilding the Brain Machine firmware with your own brainwave sequences. All of the data for the brainwave sequence is contained in the file table.h and table.h is the only file you should attempt to edit unless you feel yourself to be a competent C programmer and are willing to risk breaking the code.

If you do break the code, you can download a clean copy from:
TheEntropyWorks.googlecode.com/files/slm.zip

Note that this firmware will not run on the hardware described by Mitch Altman in MAKE #10. There are small but significant differences in how the circuit is built. The connections required are described in comments in the file slm.c.

First, let's discuss the construction of the brainwave table. It is a sequence of numerical pairs specifying an integer beat frequency from 1 to 15 beats per second and a duration in seconds. The end of the table is an entry with a beat frequency of 0.

Mitch Altman’s Original “Brain Machine”



R5 / C2 and R6 / C1 are each Low Pass Filters
for converting the square-wave output of the
microcontroller to sine-waves (more or less).

Here’s the math:
We want the cut-off frequency (Fc) of the Low Pass Filter < half 400Hz tone we’ll be using:
Fc = 1 / (2 * Pi * R * C) = 1 / (6.28 * 1000 * .000001) = 160Hz
This works because a 400Hz square-wave can be modeled as a 400Hz sine-wave
with lots of multiples of this frequency added to it.
The LPF greatly diminishes the volume of all frequencies greater than 400Hz,
leaving the 400Hz sine-wave mostly untouched.

Distributed under Creative Commons 2.5 - Attrib. & Share Alike	
TITLE: SLM for MAKE (modified from minipov3)	
Document Number: modified from original dwg at: http://www.ladyada.net/make/minipov3	REV:
Date: 1/31/2007	Sheet: 1/1

To begin the brainwave table type the following:

BW_TABLE_BEGIN
This should be followed by a number of lines that look like this:

BW_TABLE_ENTRY (14, 60)

This specifies a 14 beat per second beta frequency for 60 seconds.

At the end of the sequence include the following:

BW_TABLE_END

The brainwave table is really all you need to specify, but there are a number of other things that you can control to customize your experience.

You can specify a base frequency for the audio part of the Brain Machine. by default this is 200Hz. I have not played around with this very much, but to change it from the default enter something like the following, which will set the base frequency to 400Hz:

BASE_FREQUENCY (400)

It is also possible to customize the LEDs. This can provide some interesting effects. By default, the red LEDs flash while the blue and green LEDs fade in and out creating a moving field of color as a background.

There are a few constants already defined so that you can specify LEDs. These are:

RED: Specifies both left and right red LEDs
LEFTRED: Left red LED only
RIGHTRED: Right LED led only
GREEN: Specifies both left and right green LEDs
LEFTGREEN: Left green LED only
RIGHTGREEN: Right green LED only
BLUE: specifies Both left and right blue LEDs
LEFTBLUE: Left blue LED only
RIGHTBLUE: Right blue LED only

Then there are a number of constants that you can define using the constants above to control the behavior of the LEDs. These are:

FLASH: Chose which LEDs will flash
FADE1: These LEDs will fade in while the FADE2 LEDs fade out
FADE2: these LEDs will fade in while the FADE1 LEDs fade out

To set these constants you must use the C compiler "define" directive. By flashing the red LEDs and leaving the blue and green LEDs off, you can emulate the Brain Machine in

Make: 10. The syntax for this is:

#define FLASH RED
#define FADE1 0
#define FADE2 0

My favorite configuration is to flash the blue LEDs and have the red and green LEDs fade back and forth between the right and left eyes. This gives a less harsh flash with a warm moving background. To get this effect use the following:

#define FLASH BLUE
#define FADE1 LEFTRED | RIGHTGREEN
#define FADE2 LEFTGREEN | RIGHTRED

The vertical bar in the statements above is a logical or operand. It is used to add multiple LEDs to a channel. Adding the same LED to FADE1 and FADE2 makes no sense and will generate an error.

Adding the same LED to either FADE1 or FADE2 and to FLASH will also generate an error, but can be enabled by including the following line:

#define UNSTRICT

Here is an effect that I haven't yet tried but mean to. To have flashing LEDs that fade from red to magenta to blue with the colors cross-fading, so that when one eye LED is red, the other LED is blue, use the following:

#define UNSTRICT
#define FLASH RED | BLUE
#define FADE1 LEFTRED | RIGHTBLUE
#define FADE2 RIGHTRED | LEFTBLUE

The last element you can control is LED intensity. There are three parameters you can define:

FADEMAX: The maximum intensity for LED fades.
FADEMIN: The minimum intensity for LED fades.
FLASHINT: The intensity of the flashing LEDs.

Legitimate values for these are in the range of 0 to 127. It is possible to use values outside of this range but the LED behavior will be unpredictable. It is also possible to specify a fade minimum greater than the fade maximum; and although the behavior in this case will be predictable, I don't care to try to describe it.

So what are reasonable values? I like to set the flash intensity to the maximum (127), so that it will be visible; the fade maximum to 63 so that it doesn't wash out the flashing, and the fade minimum to 0.

Note that in the UNSTRICT example above, the lesser of flash intensity and fade maximum will be the value that is used. I would probably use 127.

Here is an example of a complete table.h file:

```
#define FLASHINT 127
#define FADEMAX 63
#define FADEMIN 0
#define FLASH RED
#define FADE1 LEFTBLUE | RIGHTGREEN
#define FADE2 LEFTGREEN | RIGHTBLUE

BASE_FREQUENCY (200)

BW_TABLE_BEGIN
  BW_TABLE_ENTRY (14, 60)
  BW_TABLE_ENTRY (11, 10)
  BW_TABLE_ENTRY (14, 20)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY (14, 15)
  BW_TABLE_ENTRY (11, 20)
  BW_TABLE_ENTRY (14, 10)
  BW_TABLE_ENTRY (11, 30)
  BW_TABLE_ENTRY (14, 5)
  BW_TABLE_ENTRY (11, 60)
  BW_TABLE_ENTRY ( 7, 10)
  BW_TABLE_ENTRY (11, 30)
  BW_TABLE_ENTRY ( 7, 20)
  BW_TABLE_ENTRY (11, 30)
  BW_TABLE_ENTRY ( 7, 30)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY ( 7, 60)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY (14, 1)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY ( 7, 60)
  BW_TABLE_ENTRY ( 2, 1)
  BW_TABLE_ENTRY ( 7, 10)
  BW_TABLE_ENTRY ( 2, 1)
  BW_TABLE_ENTRY ( 7, 10)
  BW_TABLE_ENTRY ( 2, 1)
  BW_TABLE_ENTRY ( 7, 30)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY (14, 1)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY ( 7, 30)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY (14, 1)
  BW_TABLE_ENTRY (11, 20)
```

```
  BW_TABLE_ENTRY (14, 5)
  BW_TABLE_ENTRY (11, 20)
  BW_TABLE_ENTRY (14, 15)
  BW_TABLE_ENTRY (11, 15)
  BW_TABLE_ENTRY (14, 20)
  BW_TABLE_ENTRY (11, 10)
  BW_TABLE_ENTRY (14, 25)
  BW_TABLE_ENTRY (11, 5)
  BW_TABLE_ENTRY (14, 60)
BW_TABLE_END
```

Code Comments (From Within table.h) Remixed

[**Editor Note:** The following code comments from Cossum are masterfully remixed for brevity and ease of reading, so as to *stay in context* with the above documentation]

This implementation attempts to synthesize a sine wave-like audio output more pleasing to the ear and uses RGB LEDs. The firmware is intended to blink one color while fading the others in and out, creating a moving field of color behind.

This effect was originally devised to test my code for PWM control of the LEDs, but I found it pleasing and left it in. Other effects such as blinking different colors at different beat frequencies, or having each color beat simultaneously at different frequencies should be trivial modifications to the code.

This software will run on an Arduino/ATmegaX68 (8MHz or 16MHz) with the following connections:

PB1 - base frequency audio channel
PB2 - offset frequency audio channel
PD2,PD5 - red LEDs
PD3,PD6 - green LEDs
PD4,PD7 - blue LEDs

For the Arduino, that's:

D9 - base frequency audio channel
D10 - offset frequency audio channel
D2,D5 - red LEDs
D3,D6 - green LEDs
D4,D7 - blue LEDs

It will also run on an ATtiny2313, specifically a modified Adafruit Industries MiniPOV3 kit clocked at either 8MHz or 16MHz. The connections to the ATtiny2313 should be:

PB3 - base frequency audio channel
PB4 - offset frequency audio channel

PB0,PB5 - red LEDs
PB1,PB6 - green LEDs
PB2,PB7 - blue LEDs

Note that these connections differ from Mitch's version most significantly in that one of the audio channels is moved from PB2 to PB4 so that it can be connected to OC1B output of timer 1.

The LED outputs can be moved around freely by modifying symbols defined in the file slm.h. The green and blue outputs need not really be connected. By defining the symbols RRED and LRED in the file slm.h as PB0 and PB1, one can convert an existing SLM with the only hardware modification being the movement of one audio channel.

How's It Work?

All of the outputs are generated in the timer 1 overflow interrupt service routine. The only things that happen in the main program are hardware initialization, setting up the brainwave sequence, then looping waiting for the sequence to end (while periodically changing the brightness of the green and blue LEDs), and finally stopping timer 1, turning off all the outputs and putting the MCU to sleep.

Timer 1 overflows every 512 ticks of the MCU's clock. At 8MHz this results in an interrupt frequency of 15.625KHz, at 16MHz it's 31.250KHz.

The advantage of running at 16MHz is that the audio PWM frequency will be well above the range of human hearing. At 8MHz one must rely on a low pass filter, cheap headphones and high frequency hearing loss.

At each timer interrupt, a pulse is output on OC1A with a width proportional to the desired amplitude of the base frequency sine wave. The offset frequency is output as a pulse on OC1B.

For each channel, the integer part of a 24 bit fixed point (8 bit integer, 16 bit fraction), phase accumulator is used as an index into a 256 byte table of 8 bit amplitudes. For each entry [i] in the amplitude table, sinetab[i] = 127 + sin(i * pi / 128) * 127.

For each frequency to be output, a phase step is calculated at compile time. This step is the change in phase for the frequency at each interrupt and is calculated by multiplying the frequency by the size of the sine wave table and dividing by the interrupt frequency. Step sizes for the base frequency and offset frequencies from 1 to 15 Hz above the base frequency are stored in a lookup table.

The flash rate of the LEDs is controlled by comparing the phases of the offset and base waveforms. The sign of the result of offset phase - base phase determines the state of the LEDs. Color fading is done with seven bit PWM. So the interrupt frequency of 15.625KHz is divided by 128 for a PWM frequency of ~120Hz. At every interrupt a counter is compared to 128.

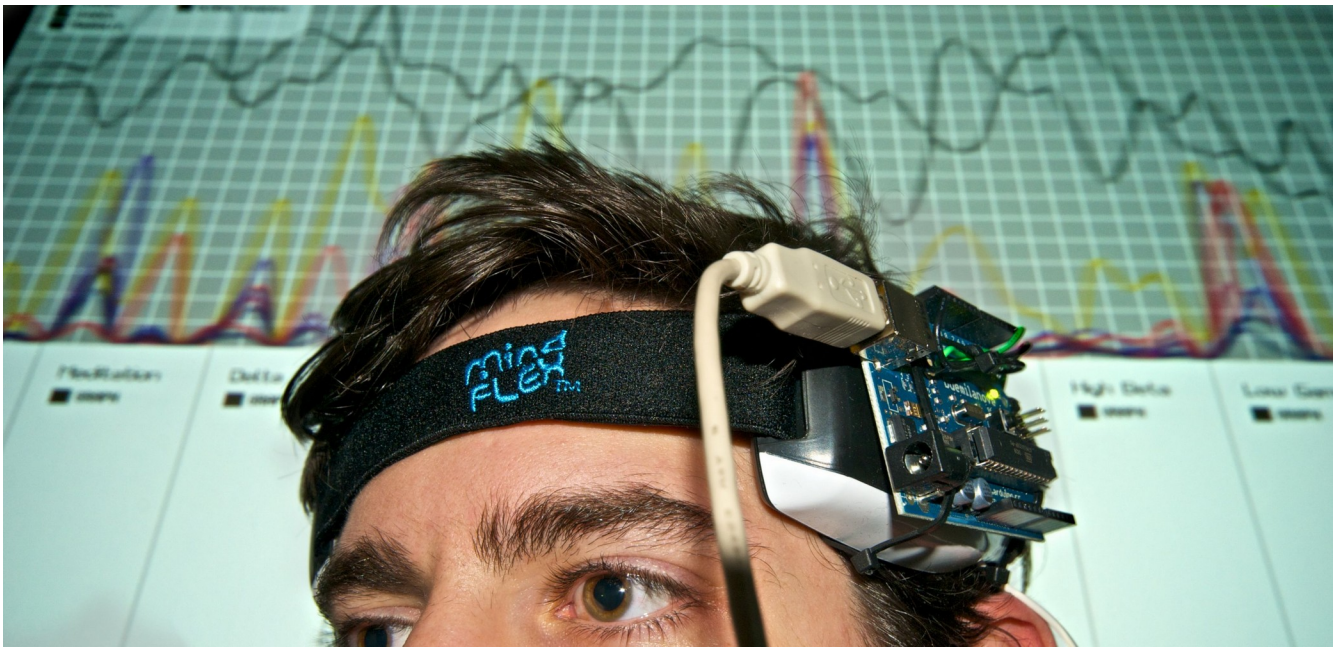
When it reaches 128, it is reset to zero, the current values of all colors are copied to locations to be used for comparisons during the current PWM period, and all the LEDs are turned on. At every interrupt, the stored color values are compared to the counter and when they are equal the corresponding LEDs are turned off. The counter is then incremented.

Pt. 4: The Art Of EEG Toy Hacking Gone Wild!

How To Hack Toy EEGs

Attribution: "*How To Hack Toy EEGs*" by direct permission from Eric Mika to specifically be included in Ultimate Arduino Brainhackar's Handbook via CC BY SA.

Eric's Home Page: EricMika.com



Arturo Vidich, Sofy Yuditskaya, and I needed a way to read brains for our Mental Block project last fall. After looking at the options, we decided that hacking a toy EEG would be the cheapest / fastest way to get the data we wanted. Here's how we did it.

The Options

A non-exhaustive list of the consumer-level options for building a brain-computer interface:

Open EEG offers a wealth of hardware schematics, notes, and free software for building your own EEG system. It's a great project, but the trouble is that the hardware costs add up quickly, and there isn't a plug-and-play implementation comparable to the EEG toys.

The Neurosky MindSet is a reasonable deal as well — it's wireless, supported, and plays nicely with the company's free developer tools. For our purposes, though, it was still a bit spendy. Since NeuroSky supplies the EEG chip and hardware for the Force Trainer and Mind Flex toys, these options represent a cheaper (if less convenient) way to get the same data.

The silicon may be the same between the three, but our tests show that each runs slightly different firmware which accounts for some variations in data output. The Force Trainer, for example, doesn't output EEG power band values — the Mind Flex does. The MindSet, unlike the toys, also gives you access to raw wave data.

However, since we'd probably end up running an FFT on the wave anyway (and that's essentially what the EEG power bands represent), we didn't particularly miss this data in our work with the Mind Flex.

Comparison At A Glance:

	Open EEG	Force Trainer	Mind Flex	MindSet
Description	Plans and software for building an EEG from scratch	Levitating ball game from Uncle Milton	Levitating ball game from Mattel	Official headset from NeuroSky
Attention / Meditation Values	No	Yes	Yes	Yes
EEG Power Band Values	Yes (roll your own FFT)	No	Yes	Yes
Raw wave values	Yes	No	No	Yes
Cost	$200+	$75 (street)	$80 (street)	$200

Given all of this, I think the Mind Flex represents a sweet spot on the price / performance curve.

It gives you almost all of the data the Mind Set for less than half the cost. The hack and accompanying software presented below works fine for the Force Trainer as well, but you'll end up with less data since the EEG power values are disabled in the Force Trainer's firmware from the factory.

Of course, the Mind Flex is supposed to be a black-box toy, not an officially supported development platform — so in order to access the actual sensor data for use in other contexts, we'll need to make some hardware modifications and write some software to help things along. Here's how.

But first, the inevitable caveat: *Use extreme caution when working with any kind of voltage around your brain, particularly when wall power is involved. The risks are small, but to be on the safe side* **you should only plug the Arduino + Mind Flex combo into a laptop running on batteries alone**.

(My thanks to Viadd for pointing out this risk in the comments.) Also, performing the modifications outlined below means that you'll void your warranty. If you make a mistake you could damage the unit beyond repair. The modifications aren't easily reversible, and they may interfere with the toy's original ball-levitating functionality.

However, I've confirmed that when the hack is executed properly, the toy will continue to function — and perhaps more interestingly, you can skim data from the NeuroSky chip without interfering with gameplay. In this way, we've confirmed that the status lights and ball-levitating fan in the Mind Flex are simply mapped to the "Attention" value coming out of the NeuroSky chip.

The Hardware

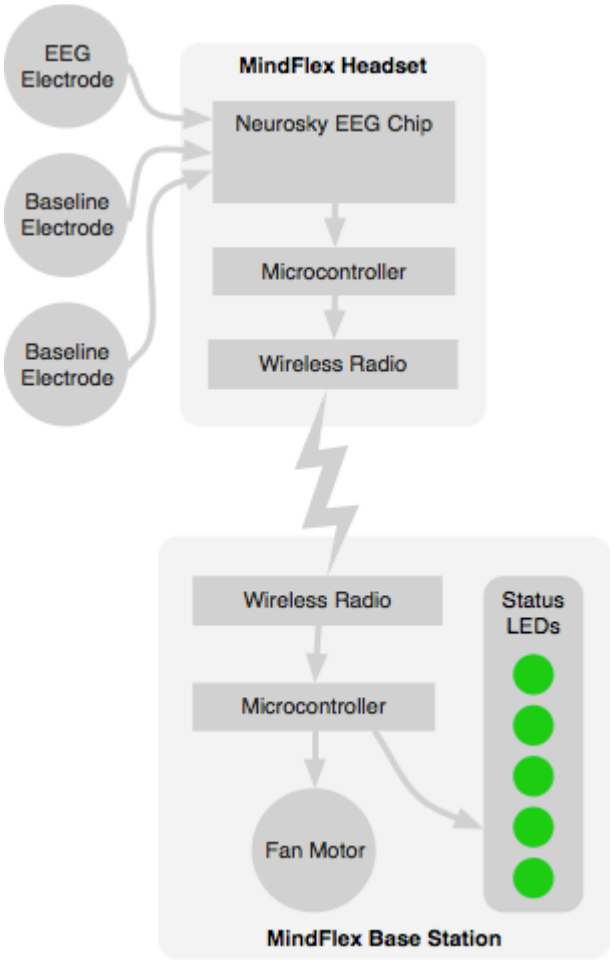
Here's the basic layout of the Mind Flex hardware. Most of the action is in the headband, which holds the EEG hardware.

A micro controller in the headband parses data from the EEG chip and sends updates wirelessly to a base station, where a fan levitates the ball and several LEDs illuminate to represent your current attention level.

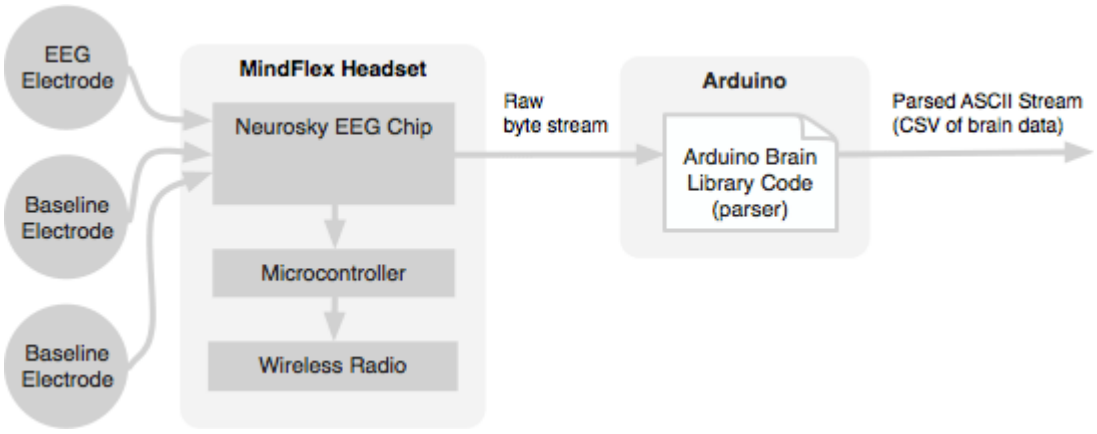
This schematic below immediately suggests several approaches to data extraction. The most common strategy we've seen is to use the LEDs on the base station to get a rough sense of the current attention level.

This is nice and simple, but five levels of attention just doesn't provide the granularity we were looking for.

A Quick Aside: *Unlike the Mind Flex, the Force Trainer has some header pins (probably for programming / testing / debugging) which seem like an ideal place to grab some data. Others have reported success with this approach. We could never get it to work.*



We decided to take a higher-level approach by grabbing serial data directly from the NeuroSky EEG chip and cutting the rest of the game hardware out of the loop, leaving a schematic that looks more like this:



The Hack

Parts List:

- 1 x Mind Flex
- 3 x AAA batteries for the headset
- 1 x Arduino (any variety), with USB cable
- 2 x 12” lengths of solid core hookup wire (around #22 or #24 gauge is best)
- A PC or Mac to monitor the serial data

Software List:

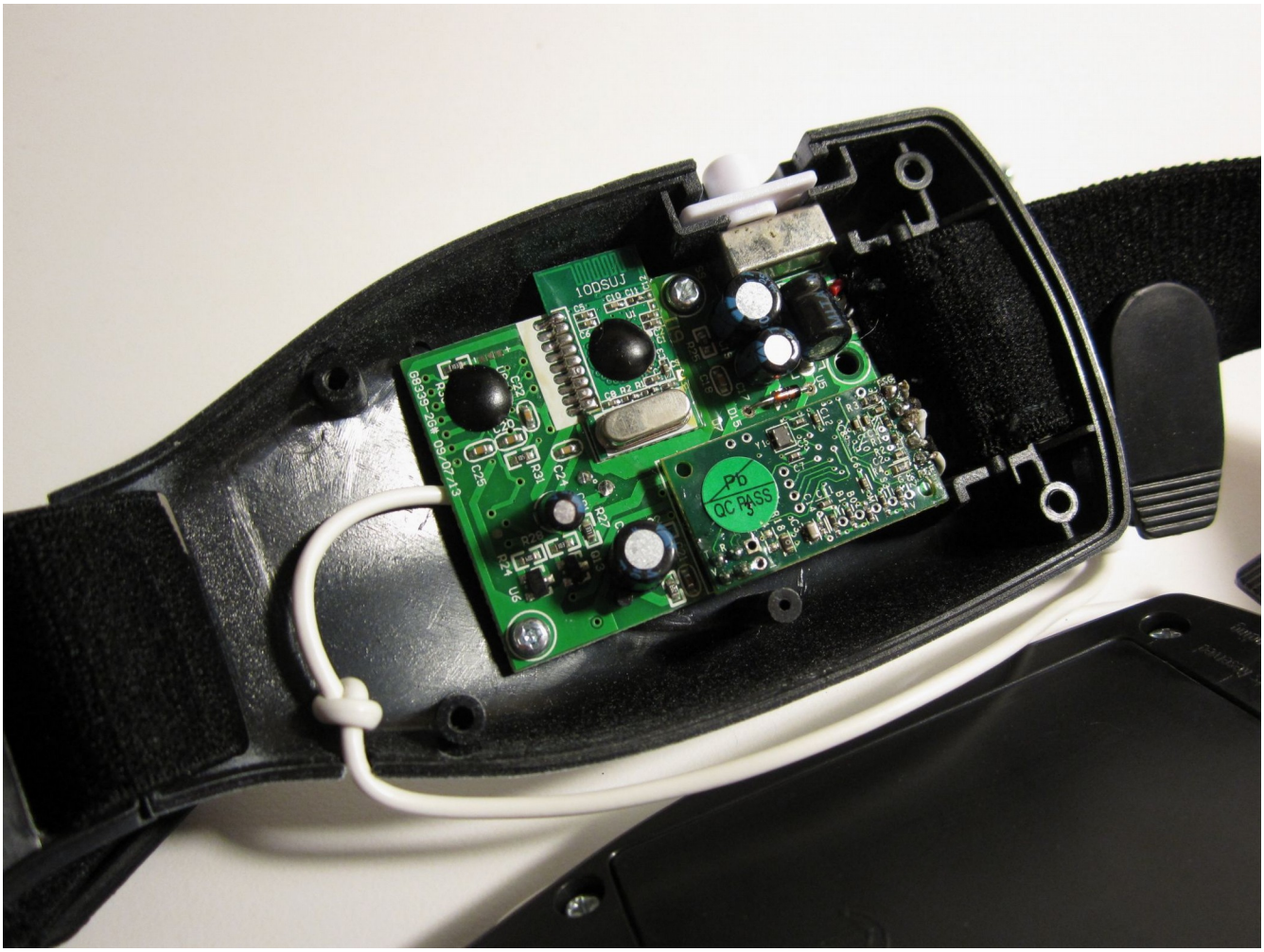
- Arduino Brain Library (download here)
- Optional: Processing Brain Visualizer (download here, it will help to have Processing as well)
- Optional (required for the visualizer): controlP5 Processing GUI Library (download here)

The video below walks through the whole process. Detailed instructions and additional commentary follow after the video: https://vimeo.com/10184668

Step-By-Step:

1. Disassembly.

Grab a screwdriver and crack open the left pod of the Mind Flex headset. (The right pod holds the batteries.)



2. The T Pin.

The NeuroSky Board is the small daughterboard towards the bottom of the headset.

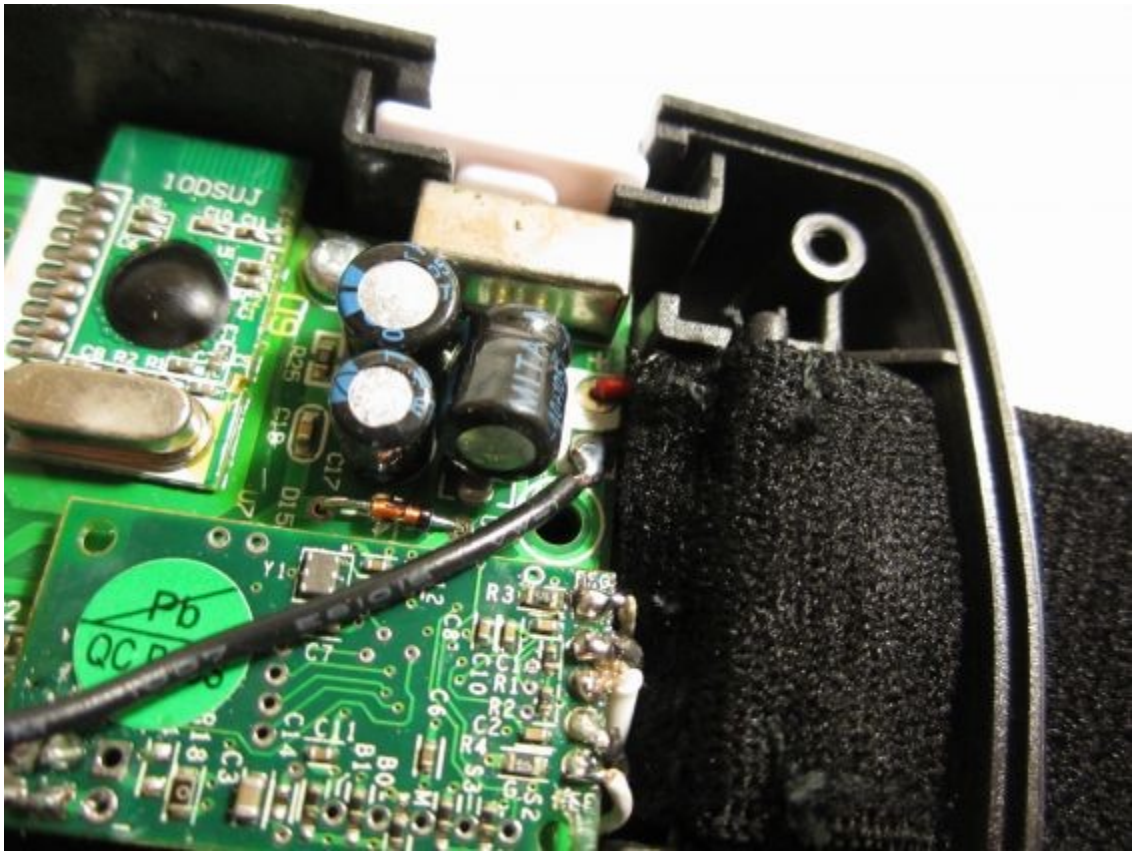
If you look closely, you should see conveniently labeled **T** and **R** pins — these are the pins the EEG board uses to communicate serially to the microcontroller on the main board, and they're the pins we'll use to eavesdrop on the brain data.

Solder a length of wire (carefully) to the "T" pin. Thin wire is fine, we used #24 gauge. Be careful not to short the neighboring pins.

3. Common ground.

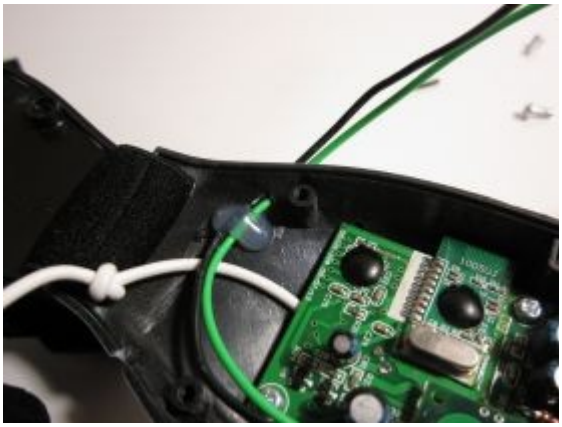
Your Arduino will want to share ground with the Mind Flex circuit. Solder another length of wire to ground — any grounding point will do, but using the large solder pad where the battery's ground connection arrives at the board makes the job easier.

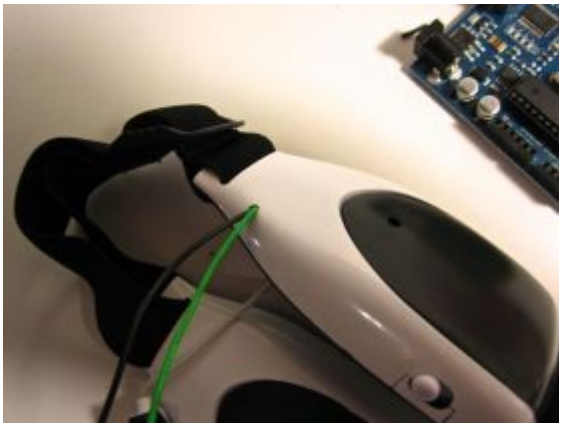
A note on power: *We've found the Mind Flex to be inordinately sensitive to power… our initial hope was to power the NeuroSky board from the Arduino's 3.3v supply, but this proved unreliable. For now we're sticking with the factory configuration and powering the Arduino and Mind Flex independently.*



4. Strain relief and wire routing.

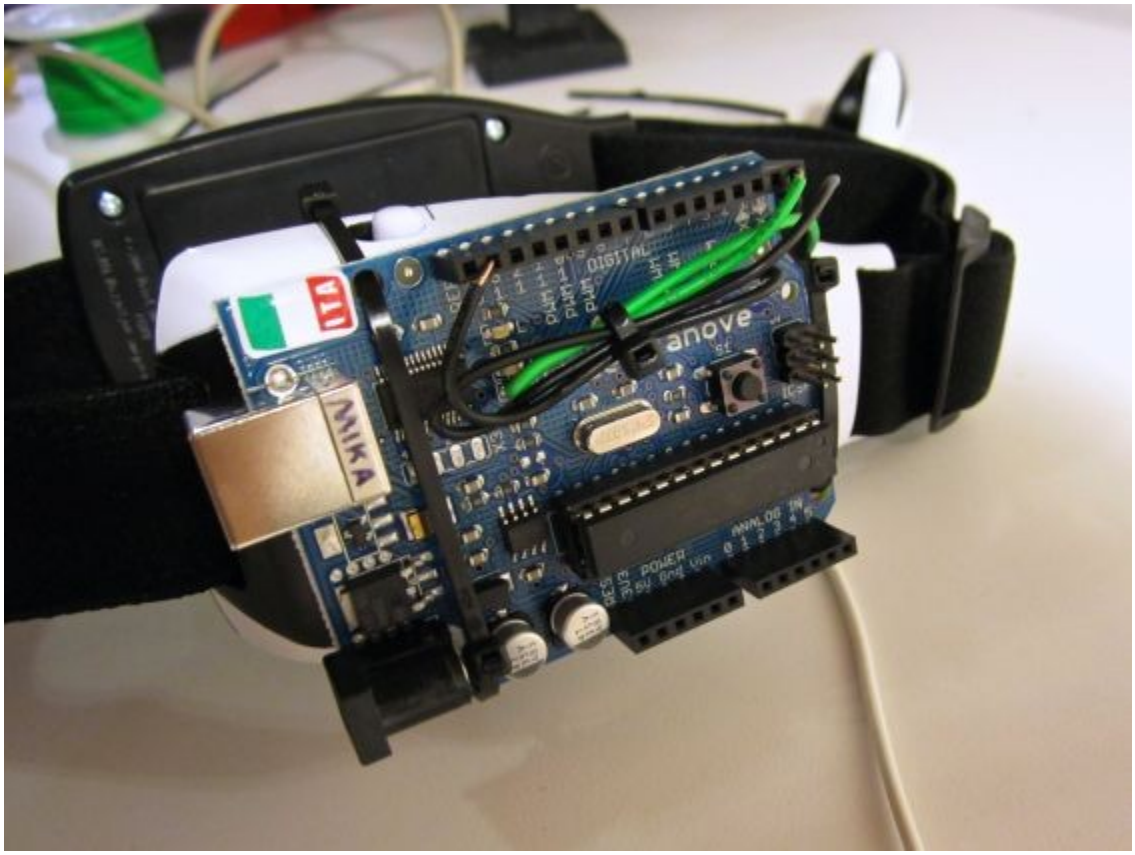
We used a dab of hot glue to act as strain relief for the new wires, and drilled a hole in the case for the two wires to poke through after the case was closed. This step is optional.





5. Hook up the Arduino.

The wire from the Mind Flex's "T" pin goes into the Arduino's **RX pin**. The ground goes… to ground. You may wish to secure the Arduino to the side of the Mind Flex as a matter of convenience. (We used zip ties.)



That's the extent of the hardware hack. Now on to the software.

The data from the NeuroSky is not in a particularly friendly format. It's a stream of raw bytes that will need to be parsed before they'll make any sense. Fate is on our side: the

packets coming from the Mind Flex match the structure from NeuroSky's official Mindset documentation. (See the *mindset_communications_protocol.pdf* document in the Mindset developer kit if you're interested.)

You don't need to worry about this, since I've written an Arduino library that makes the parsing process as painless as possible. Essentially, the library takes the raw byte data from the NeuroSky chip, and turns it into a nice ASCII string of comma-separated values.

6. Load up the Arduino.

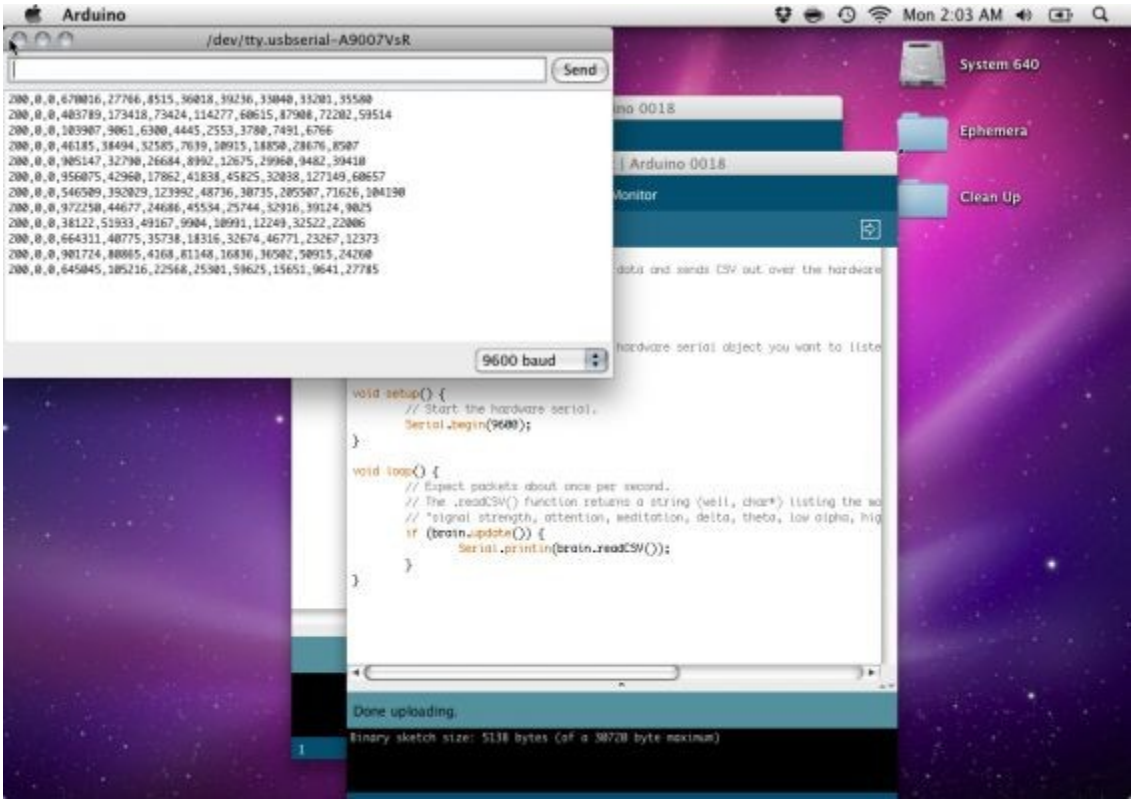
Download and install the Arduino Brain Library — it's available here. Open the *BrainSerialOut* example and upload it to your board. (You may need to disconnect the RX pin during the upload.)

The example code looks like this:

```
#include <Brain.h>

// Set up the brain parser, pass it the hardware serial object you want to
listen on.
Brain brain(Serial);

void setup() {
        // Start the hardware serial.
        Serial.begin(9600);
}

void loop() {
        // Expect packets about once per second.
        // The .readCSV() function returns a string (well, char*) listing the
most recent brain data, in the following format:
        // "signal strength, attention, meditation, delta, theta, low alpha,
high alpha, low beta, high beta, low gamma, high gamma"
        if (brain.update()) {
                Serial.println(brain.readCSV());
        }
}
```

7. Test.

Turn on the Mind Flex, make sure the Arduino is plugged into your computer, and then open up the Serial Monitor. If all went well, you should see the following:



Here's how the CSV breaks down: "signal strength, attention, meditation, delta, theta, low alpha, high alpha, low beta, high beta, low gamma, high gamma"

(More on what these values are supposed to mean later in the article. Also, note that if you are hacking a Force Trainer instead of a Mind Flex, you will only see the first three values — signal strength, attention, and meditation.)

If you put the unit on your head, you should see the "signal strength" value drop to 0 (confusingly, this means the connection is good), and the rest of the numbers start to fluctuate.

8. Visualize.

As exciting as the serial monitor is, you might think, "Surely there's a more intuitive way to visualize this data!" You're in luck: I've written a quick, open-source visualizer in Processing which graphs your brain activity over time (download). It's designed to work with the *BrainSerialOut* Arduino code you've already loaded.

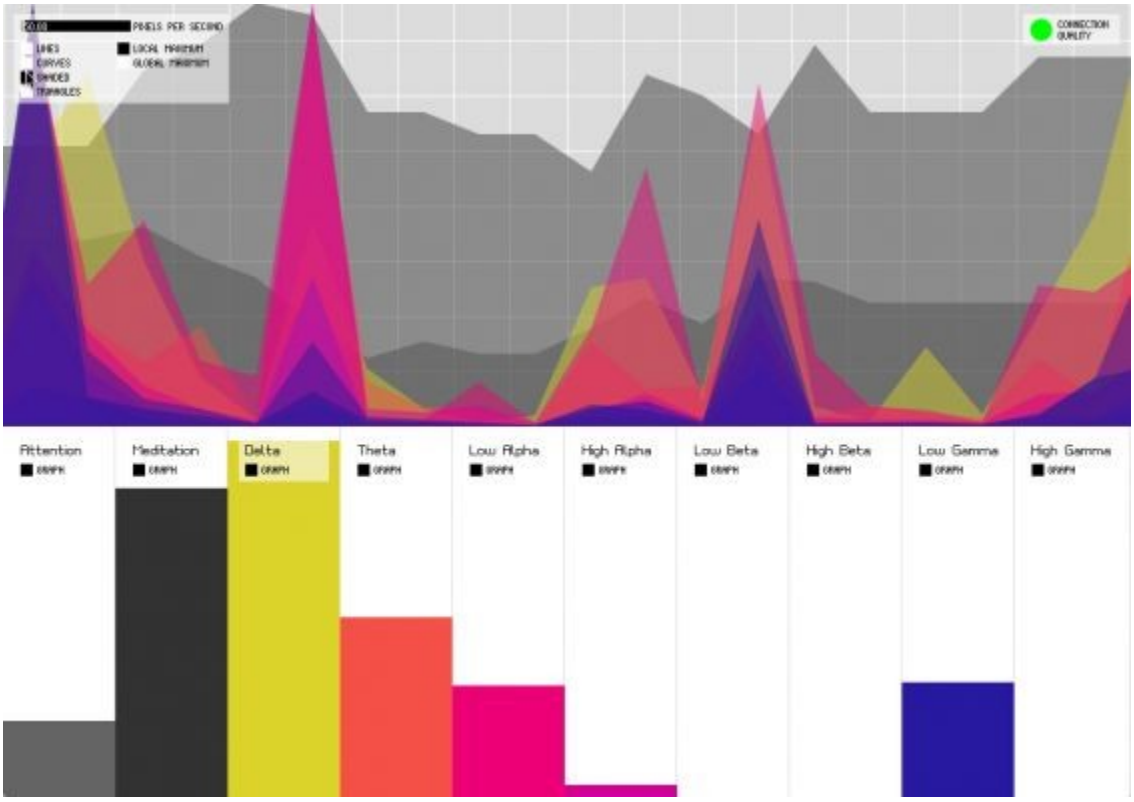
Download the code, and then open up the *brain_grapher.pde* file in Processing.

With the Mind Flex plugged in via USB and powered on, go ahead and run the Processing sketch. (Just make sure the Arduino IDE's serial monitor is closed, otherwise Processing won't be able to read from the Mind Flex.)

You may need to change the index of the serial list array in the brain_grapher.pde file, in case your Arduino is not the first serial object on your machine:

```
serial = new Serial(this, Serial.list()[0], 9600);
```

You should end up with a screen like this:



About The Data

So what, exactly, do the numbers coming in from the NeuroSky chip mean?

The Mind Flex (but not the Force Trainer) provide eight values representing the amount of electrical activity at different frequencies. This data is heavily filtered / amplified, so where a conventional medical-grade EEG would give you absolute voltage values for each band, NeuroSky instead gives you relative measurements which aren't easily mapped to real-world units.

A run down of the frequencies involved follows, along with a grossly oversimplified summary of the associated mental states.

- Delta (1-3Hz): sleep
- Theta (4-7Hz): relaxed, meditative

- Low Alpha (8-9Hz): eyes closed, relaxed
- High Alpha (10-12Hz)
- Low Beta (13-17Hz): alert, focused
- High Beta (18-30Hz)
- Low Gamma (31-40Hz): multi-sensory processing
- High Gamma (41-50Hz)

In addition to these power-band values, the NeuroSky chip provides a pair of proprietary, black-box data values dubbed “attention” and “mediation”.

These are intended to provide an easily-grokked reduction of the brainwave data, and it’s what the Force Trainer and Mind Flex actually use to control the game state. We’re a bit skeptical of these values, since NeuroSky won’t disclose how they work, but a white paper they’ve released suggests that the values are at least statistically distinguishable from nonsense.

Here’s the company line on each value:

- **Attention:**

  Indicates the intensity of a user’s level of mental “focus” or “attention”, such as that which occurs during intense concentration and directed (but stable) mental activity. Distractions, wandering thoughts, lack of focus, or anxiety may lower the Attention meter levels.

- **Meditation:**

  Indicates the level of a user’s mental “calmness” or “relaxation”.

  Meditation is related to reduced activity by the active mental processes in the brain, and it has long been an observed effect that closing one’s eyes turns off the mental activities which process images from the eyes, so closing the eyes is often an effective method for increasing the Meditation meter level.

  Distractions, wandering thoughts, anxiety, agitation, and sensory stimuli may lower the Meditation meter levels.

At least that’s how it’s supposed to work. We’ve found that the degree of mental control over the signal varies from person to person. Ian Cleary, a peer of ours at ITP, used the Mind Flex in a recent project. He reports that about half of the people who tried the game were able to exercise control by consciously changing their mental state.

The most reasonable test of the device's legitimacy would be a comparison with a medical-grade EEG. While we have not been able to test this ourselves, NeuroSky has published the results of such a comparison.

Their findings suggest that the the NeuroSky chip delivers a comparable signal. Of course, NeuroSky has a significant stake in a positive outcome for this sort of test.

And there you have it. If you'd like to develop hardware or software around this data, I recommend reading the documentation that comes with the brain library for more information — or browse through the visualizer source to see how to work with the serial data.

If you make something interesting using these techniques, I'd love to hear about it.

March 2013 Update:

Almost three years on, I think I need to close the comments since I don't have the time (or hardware on hand) to keep up with support. Please post future issues on the GitHub page of the relevant project:

Arduino Brain Library
https://github.com/kitschpatrol/Arduino-Brain-Library

Processing Brain Grapher
https://github.com/kitschpatrol/Processing-Brain-Grapher

Most issues I'm seeing in the comments seem like the result of either soldering errors or compatibility-breaking changes to the Processing and Arduino APIs.

I'll try to stay ahead of these on GitHub and will be happy to accept pull requests to keep the code up to date and working. Thanks everyone for your feedback and good luck with your projects.

Visualizing Mind Activity With A Hacked Toy EEG

Attribution: By Darren Mothersele via CC BY SA

Hardware

The MindFlex toy includes a headset that reads EEG wave data. There's lots of information on hacking this toy to create a simple EEG machine using the limited data the headset provides by default. In this project I have extracted the full RAW EEG data using the same toy headset from the MindFlex game.

Friendly Warning And Disclaimer: Hack your toys at your own risk. I'm not responsible for any damage you do to yourself or your belongings. This information is provided here for information and education only. If you use the original hack and just use the data available in normal mode then the MindFlex still works as a game as originally intended. If you enable RAW EEG output then you break the toy!

Reading Brainwaves

An obvious place for us to start on this project might have been with the excellent OpenEEG project, but while being open, it still has substantial costs and complexity in getting up and running. For this project we looked for a simpler approach.

EEG technology has been incorporated into various toys and household items before. There are various commercial options available based on NeuroSky's ThinkGear chip.

The original Frontier Nerds blog post from 2010 provides a great starting point, but I've made a few changes along the way, including successfully extracting the RAW EEG values from the MindFlex toy.

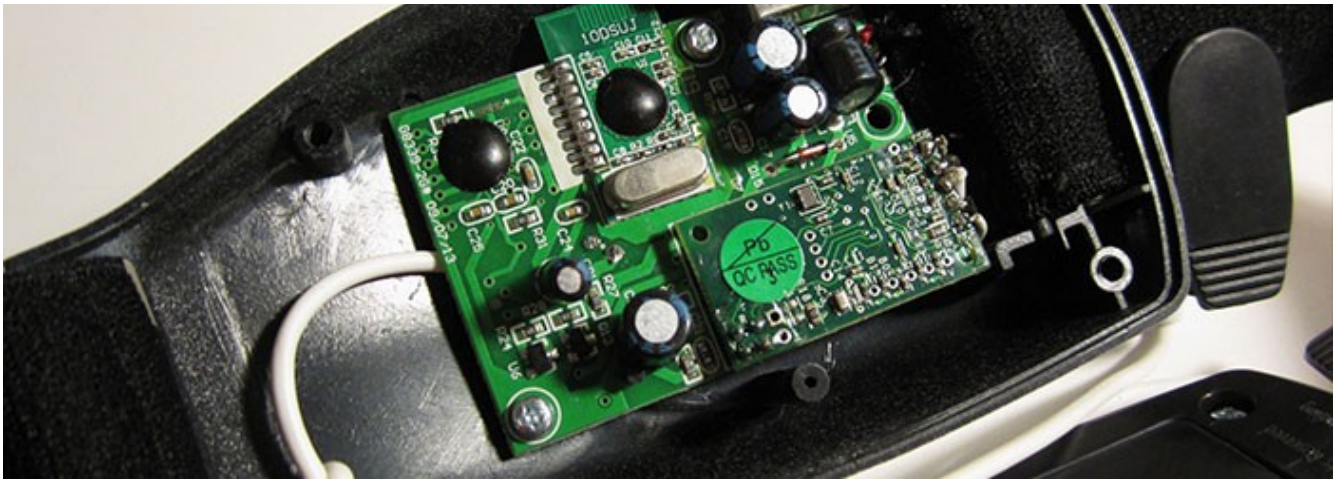
Budget EEG With The TGAM1

The TGAM1 is the controller board responsible for processing the EEG signals in all of the products based on the NeuroSky ThinkGear chip.

NeuroSky no longer sell the boards in small quantities (I've had confirmation of this from their business development team that this is due to support resources) and instead recommend that you buy the BrainWave starter kit.

This is probably a good idea if you want easy access to EEG data, but they are still over 100 GBP to buy in the UK, and the toys are available for only 40.

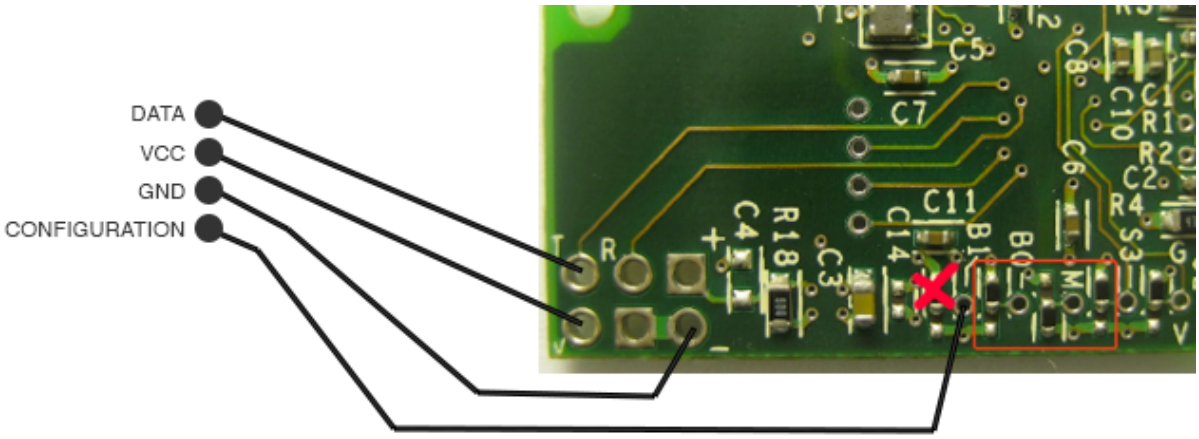
When you open the left compartment on the MindFlex you will see this:



The small board located on the top at the bottom right is the TGAM1 board.

The basic hack needs *just two solder points*. You solder on to the **T** pin to get the data out, and another connection for a common ground between your Arduinos circuit and the TGAM1.

I have also soldered two more connections on, which I will explain later.



Simple EEG Reading

The basic data that comes out of the MindFlex headset by default actually gives us quite a lot of information for only £40. In the default configuration you have soldered on only two connections, one for a common ground and one for the serial data stream.

I connect this to a SoftwareSerial pin on the Arduino and read in the packets. Here's what data you get:

- **Signal strength (0 - 200)**, where 0 means good signal, and 200 indicates a problem with the connection.

- **Attention and Meditation eSense values**, which are a proprietary measure that is calculated by the TGAM1 using NeuroSky's own algorithm. The toys based on this technology use the attention measure.

- **EEG power band values,** which are a measure of the various power bands that has been pre-calculated by the chip, this updates once per second.

The Arduino Brain Library already includes functionality for working with these values.

TGAM1 Region Change

If you bought the EU version and you're in EU then skip this bit.

The MindFlex, MindSet, Necomimi, etc all based on the TGAM1 that includes a notch filter on the EEG wave to remove electrical noise from the data.

If you bought a MindFlex from a different region then you will need to adapt it to work for you. For example, if you've bought a headset from the US it will have a notch filter at 60Hz. In Europe our "mains hum" is at 50Hz. You can switch the notch filter to work at the correct frequency with a bit of soldering.

Getting Full RAW EEG Data

Now on to getting the full raw EEG data from the MindFlex headset. In addition to the two connections you soldered on before there are two extra connections required.

This allows you to configure the TGAM1 board that is in the MindFlex to switch between two modes. The 'configure' connection is actually called B1 on the TGAM1 board. In the MindFlex headset B1 is connected to GND via a 10K resistor. I have marked this resistor with an X on the image above.

- You need to remove this 10K resistor from the circuit board, and connect B2 to VCC via a 10K resistor to enable full raw EEG data output.

- The full EEG data is output at the higher baud rate of 57600. You also get the same packets as in normal mode.

You can switch back to normal mode, with just the basic data output at baud rate of 9600 by connecting B1 back to GND with the 10K resistor instead of the VCC connection.

Signal Strength

The normal data and raw data modes both give you the summary packet once every second.

In this is a measure of *signal strength*. It should be zero to indicate that the board is getting good measurements. Anything >0 indicates poor signal, and 200 indicates a problem.

When I had the Arduino hooked up to the computer serial port I struggled to get the signal strength to 0. When running off batteries (I have an Arduino Pro mini connected to the headset's battery pack) I have had no issues with signal strength.

The issues were resolved by adding an extra GND connection from the Arduino and connecting to the body of the person being tested. However, I don't recommend this approach; running the headset *completely off batteries* is recommended.

This requires no wired connection to the computer, hence the RF link that has been implemented.

RF Data Link

I am using a cheap low-power RF transceiver to send the EEG packets to the computer. This makes the headset wireless and solves the signal strength problem that required the extra ground connection.

Receiving The Data

The Processing Brain Grapher is a good starting point to check everything is working, *but it only works when the headset is in normal mode and sending the data via the serial port*.

I am working on a Processing library that will work with the updated headset in full EEG mode; this giving access to the eSense meters, the EEG power band values, and the full EEG raw data.

I'll be expanding this section as I make more progress with the libraries and visualisations.

September 2013 Update: I've posted an update to this blog, where I provide details of how to extract the full RAW EEG data from the MindFlex headset, covered below.

Full RAW EEG Data From The MindFlex Headset Hack

Attribution: By Darren Mothersele via CC BY SA

Hardware

There have been many hacks posted online over the past few years that show how to build a simple EEG machine using the Mattel MindFlex headset.

It's simple to interface this with the Arduino to read the basic single signal EEG data the toy produces. The toy actually only uses one of the measures (attention) but it outputs various useful bits of information.

All of this is detailed in the original Frontier Nerds blog post and on our Brain Wave Visualisation project page.

The original hacks are all based around the preprocessed data that the TGAM1 board in the MindFlex toy produces. I have successfully modified one of these headsets to get **full raw EEG data** from it.

I'm in the process of writing a library and updating the project page with full details, but in the meantime here's how it works.

Full Raw EEG Values

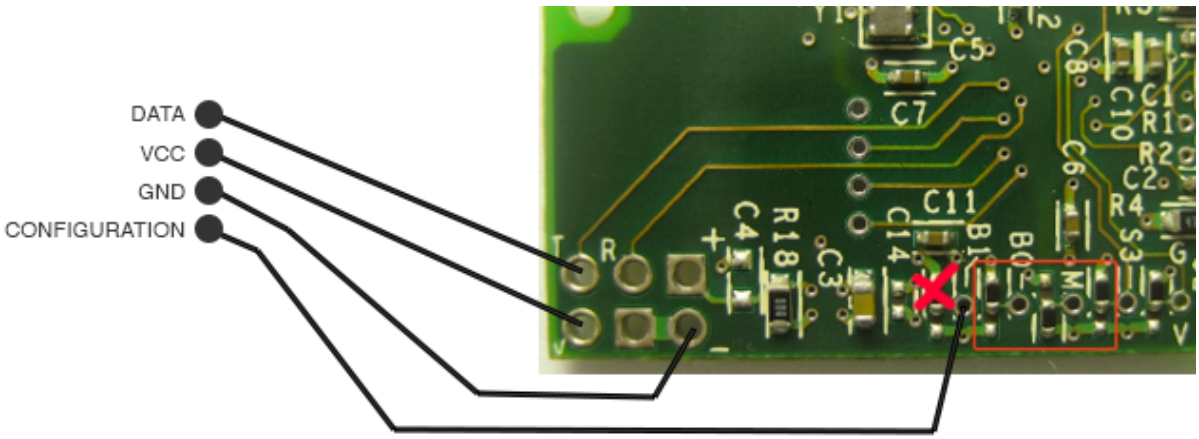
The TGAM1 board that is inside the MindFlex can run in three modes. One of these is 9600 bits per second normal mode which is how the MindFlex is configured by default.

This outputs a packet of data approximately every second that includes signal strength, attention level, meditation level, and pre-calculated values for 8 EEG power bands.

Friendly Warning and Disclaimer: Hack your toys at your own risk. I'm not responsible for any damage you do to yourself or your belongings. This information is provided here for information and education only.

If you use the original hack and just use the data available *in normal mode* then the MindFlex still works as a game as originally intended. If you enable RAW EEG output then you **break the toy**!

To enable RAW EEG you need to remove a 10K resistor from the TGAM1 board, and solder an extra connection to the board, as shown in this image:



The Hack

- Remove the 10K resistor that appears where the red X is on the photo.

You should double check with a multi-meter that you're removing the correct part. It is connected to the B1 port. This B1 port forms part of the configuration of the TGAM1 board.

In the MindFlex this 10k resistor connects the B1 port to GND which configures it in normal mode at 9600 bps.

- Solder a connection on to the B1 port, then you can add a new 10K resistor that connects the B1 port to V_{CC}. This enables raw output mode at 57600 bps.

You connect to the Arduino in the same way, *but you can't use the Arduino Brain Library with this configuration*. I'm working on an update for this.

The TGAM1 chip still sends the summary data packets that you get in normal mode once per second, but now you will also get a packet containing the raw EEG reading **512 times per second**, approximately once every 2ms.

Hacking MindWave Mobile

Attribution: Based on the SparkFun tutorial "Hacking MindWave Mobile" originally written by Sophi Kravitz via CC BY SA, and masterfully remixed by Mark Maffei.

This tutorial will explore the inner-workings of hacking the NeuroSky MindWave Mobile.

Project Scope

In this project, the analog brainwaves enter a processing ASIC chip and have digital values that are communicated over Bluetooth. I accessed only the digital data.

It is certainly possible to look at the analog brain waves before they enter the processing ASIC, but it would be much more difficult, requiring a schematic and specialized oscilloscope probes.

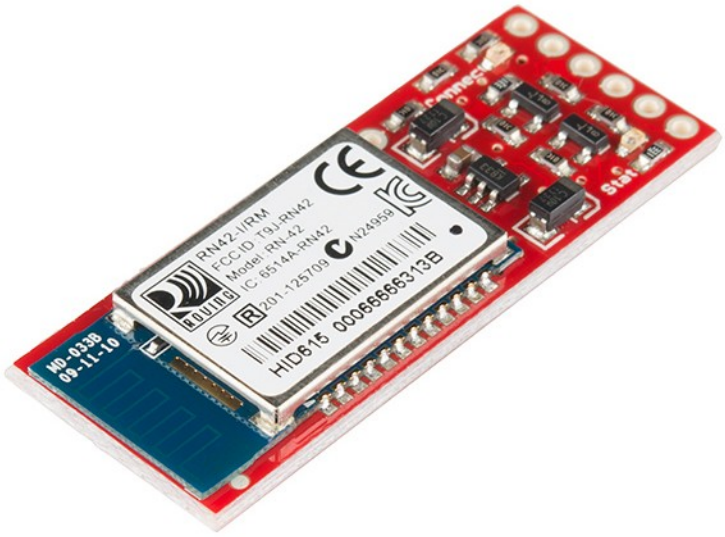
The Mindwave Mobile was chosen because it uses Bluetooth, making it easier to interface it with a microcontroller or other hardware.

Hardware

To interface with the MindWave, the RN-42 Bluetooth module was chosen. For this project, I created a custom PCB.

However, you could also use a BlueSMiRF or a Bluetooth Mate. The Bluetooth module will be connected to an Arduino Uno to read in the data being transmitted wirelessly. Once you've decided on which hardware you'll be using, connect everything.

The SparkFun Bluetooth Basics can come in quite handy if you're new to this.



The BlueSMiRF Silver module from SparkFun

Software

To read data from and configure the RN42 Bluetooth module:

- X-CTU, CoolTerm, or another serial terminal program of your choice.
- RS232 Port Logger

If unfamiliar with serial terminal emulators, check out SparkFun's terminal tutorial.

Firmware

- Here is the Arduino code.
- Here is the Processing code (interprets the MindWave data coming into the Arduino and gives you a visual representation of that data).

Sophi sporting her MindWave

Configuring The Bluetooth Module

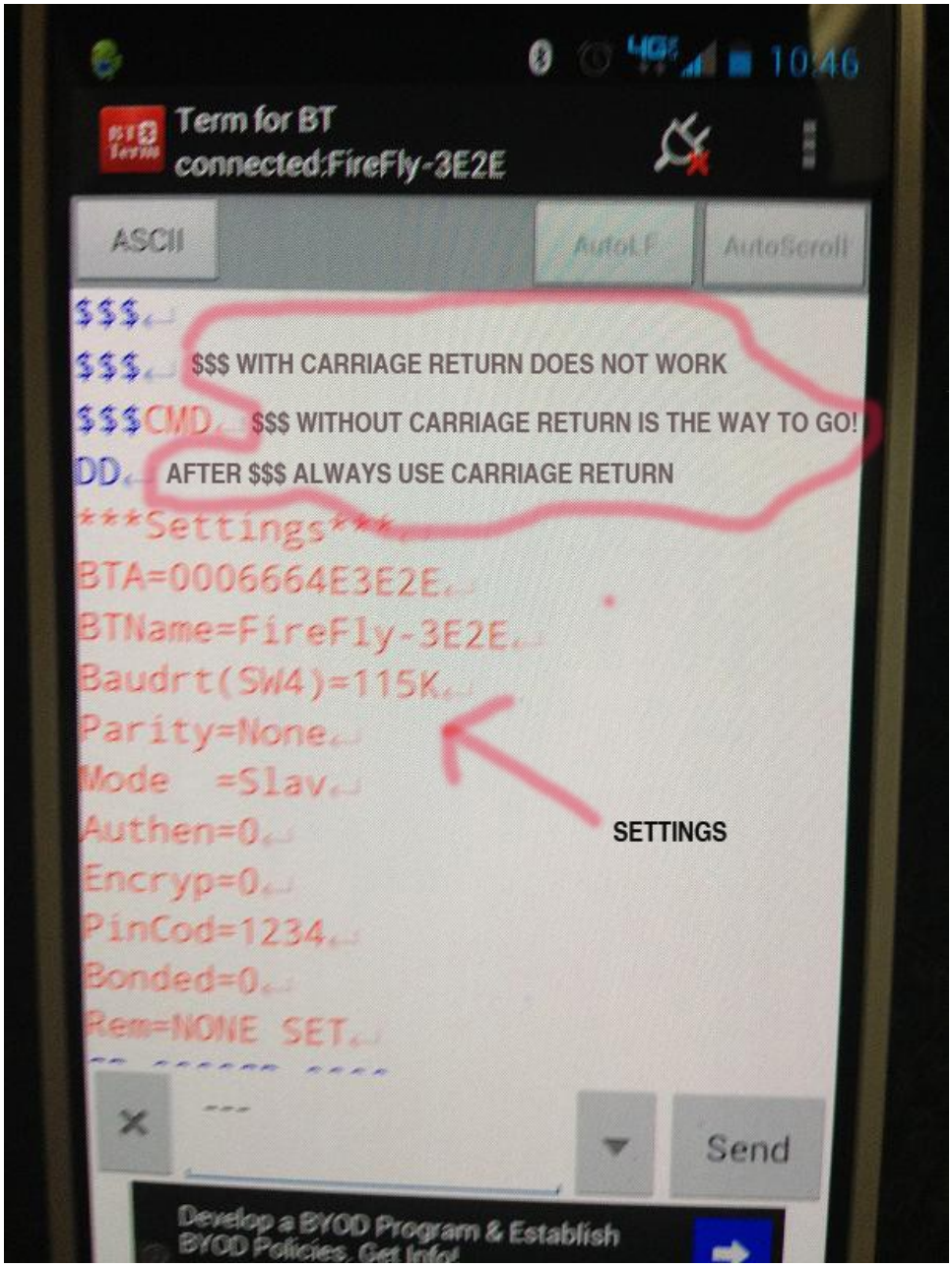
The RN42 module can be configured to behave in a variety of different ways. If you have never configured a RN Bluetooth module, scope out SparkFun's BlueSMiRF Hookup Guide.

You can use any serial terminal you wish to configure the Bluetooth module. For this project, I used my Android phone with an app called S2 Terminal for Bluetooth to configure the RN42. The configuration process went as so:

Note: If you are using the S2 Terminal program, you will need to type in ASCII and put a return after each command. Also, you only have 60 seconds after power up to enter the first command.

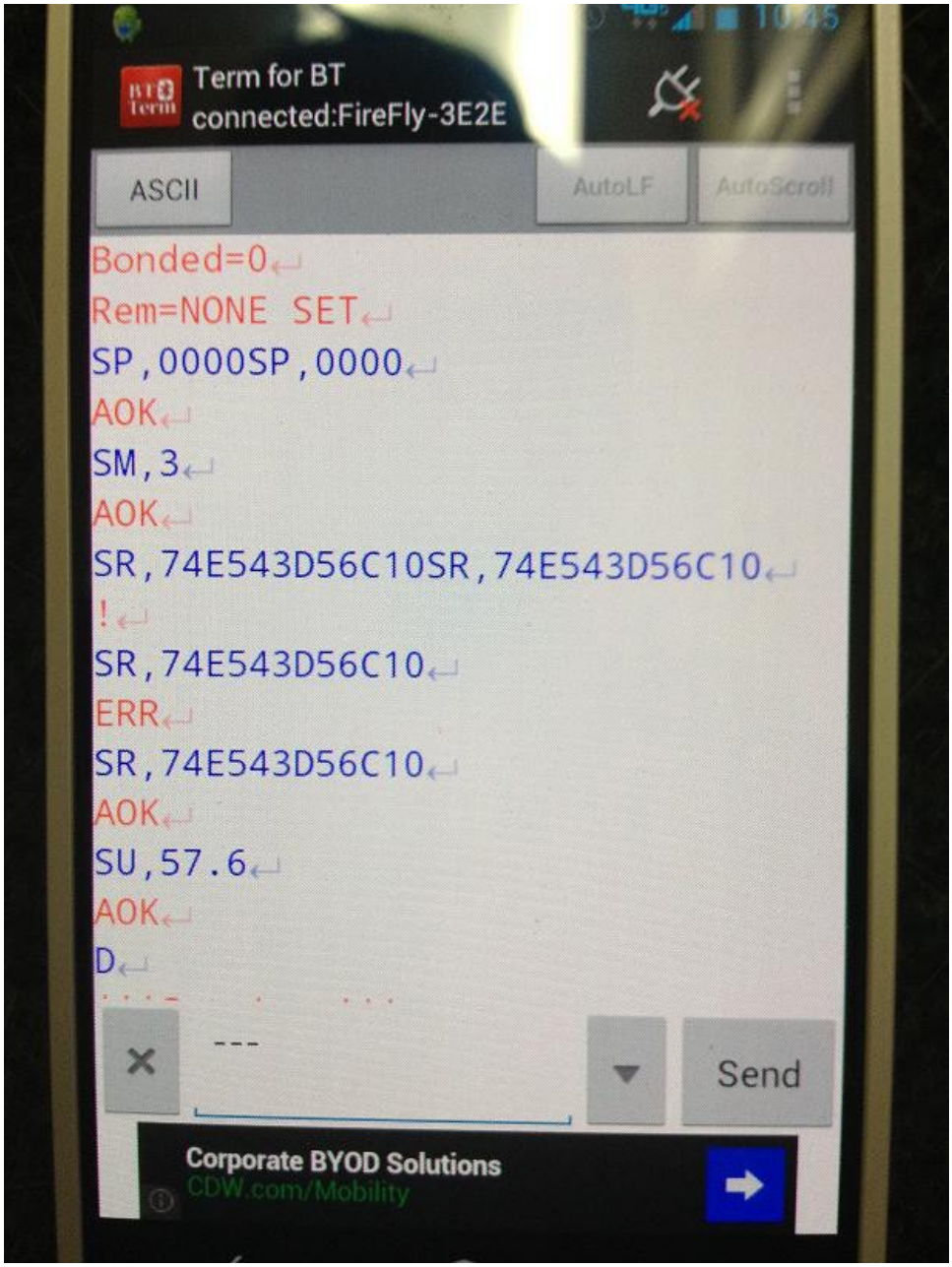
First:

1. To put the RN42 into COMMAND mode, type $$$ (NO carriage return). If successful, you will see "CMD".
2. Type: `D` + carriage return. You will see the current configuration of the RN42.



Next:

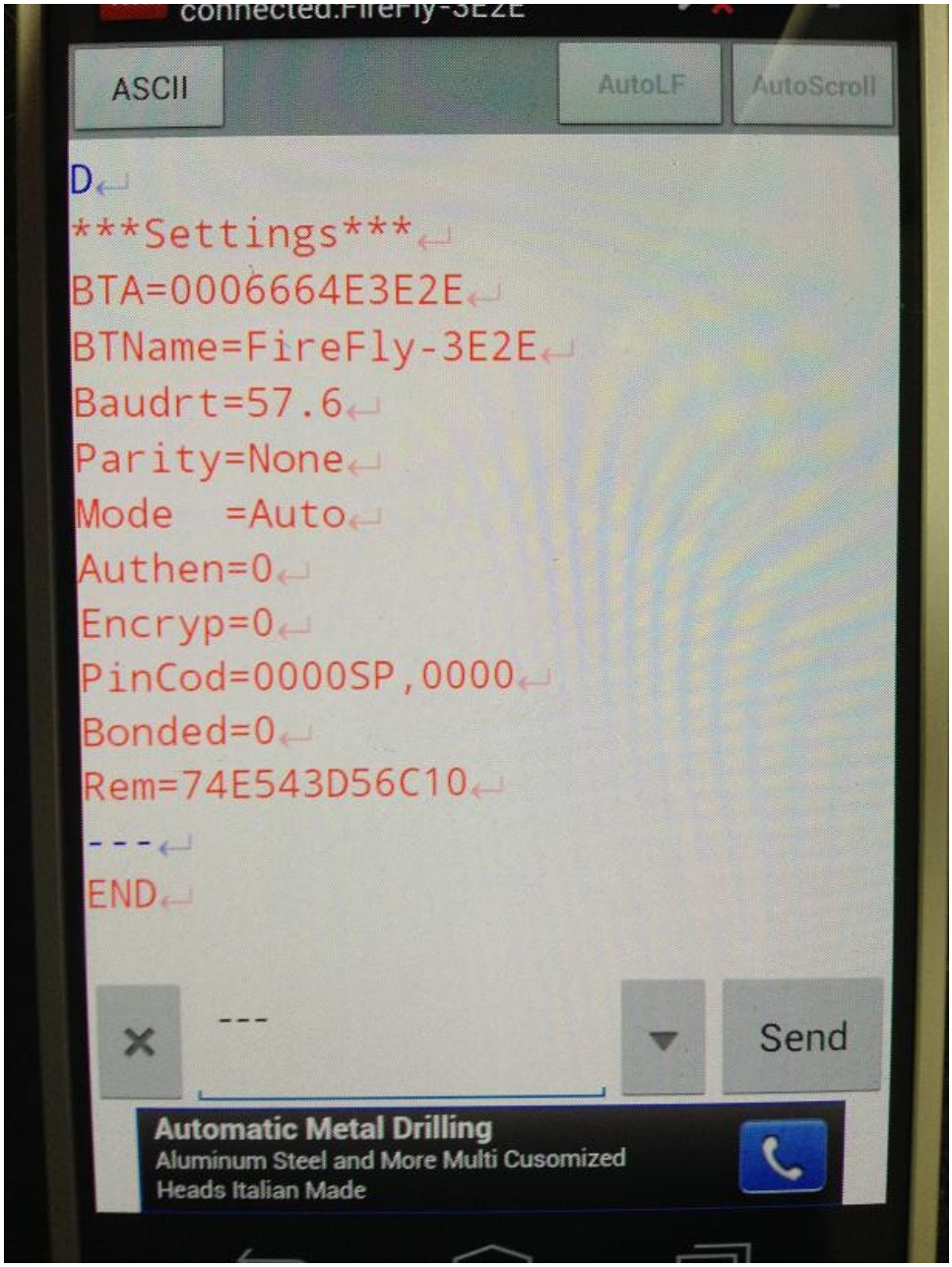
3. Type: `SP,0000` + carriage return. This will change the pincode from '1234' to '0000'. You will see "AOK" if this is done properly.
4. Type: `SM, 3` + carriage return. This will configure the RN42 to Auto-Connect Mode. Once the module is powered up, it will immediately look to connect (pair).You will see "AOK" if this is done properly.
5. Type: `SR,MAC ADDRESS` + carriage return. Insert the 12 digit address you copied from the MindWave Mobile. Look for AOK. If you don't see AOK, you'll have to retype the MAC address command.
6. Now type: `SU,57.6` + carriage return. This will change the baud rate from 115200 to 57600.



Then:

---

7. Type `D` + carriage return. Double check that the stored address is the Mac address you entered in step 5 and that it's configured to Auto, not Slave.
8. Type: `---` (three minus signs) + carriage return You should then see END.



Finally:

Turn the MindWave on. The blue light flashes for a few moments, then it will pair with your hardware. You'll know that the MindWave Mobile is paired because the blue light on the MindWave Mobile is solid.

If it is solid red, it is out of battery (AAA, lasts for about 8 hours).

Deciphering the MindWave Data

Using an FTDI basic, I was able to view the data coming in from the Arduino through my USB port. I used X-CTU to read the data.

The Arduino code linked above is mostly a combination of the Mindwave Mobile example code and Shane Clements' massaging. The code is set up to print all of the digital data available out of the MindWave Mobile. The data runs through the software serial port and is read easily on the X-CTU screen.

You can see unitless values for Delta, Theta, Alpha, Low Beta, High Beta and Gamma waves.

The MindWave's ASIC also presents a calculated value for both Attention and Meditation.

A number between 20 and 100 for Attention is normal, and a value above 40 means you are concentrating. I was able to get repeatable values with the Attention value by counting backwards from 100 to1.

We had a Bring A Hack dinner in Boulder, CO, and a few other people were able to make the Attention value rise by counting backwards from 100 to 1 also. What is interesting is that the 5 types of brainwaves measured and reported originate from different parts of the brain. They also have extremely low amplitudes and frequencies.

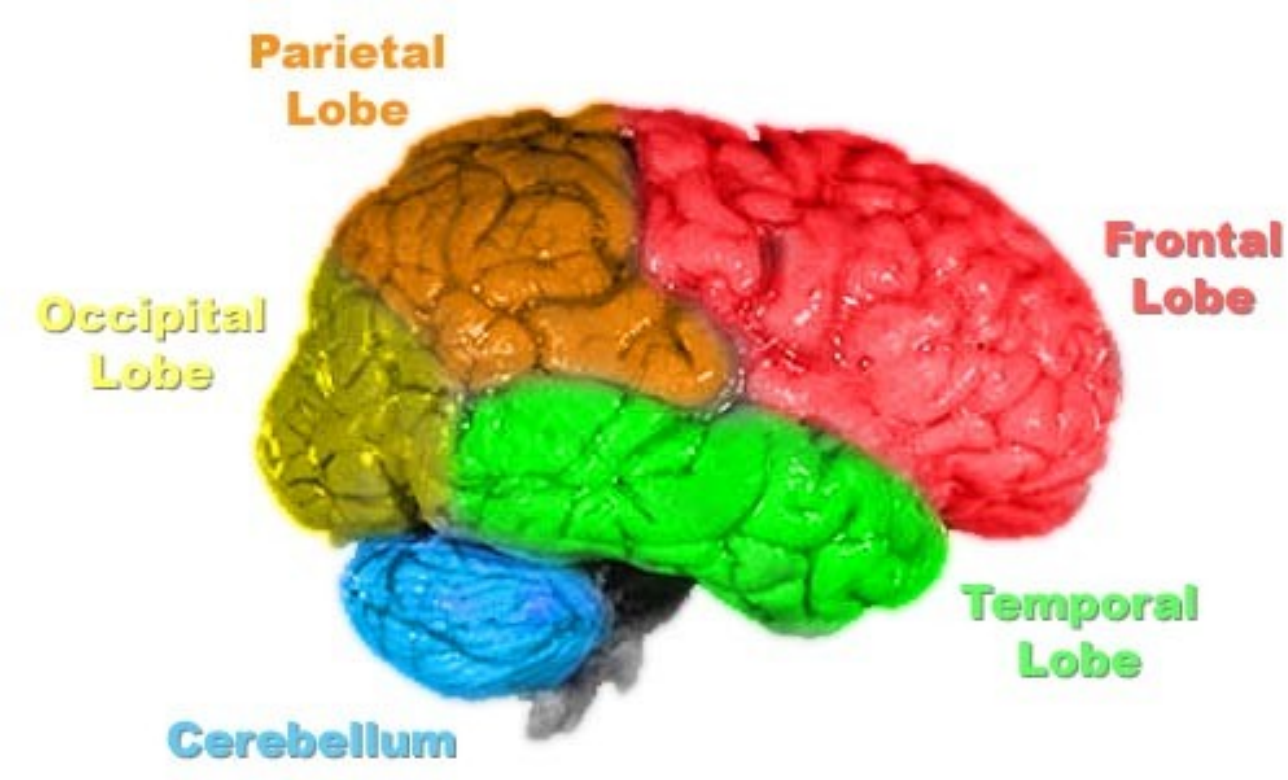
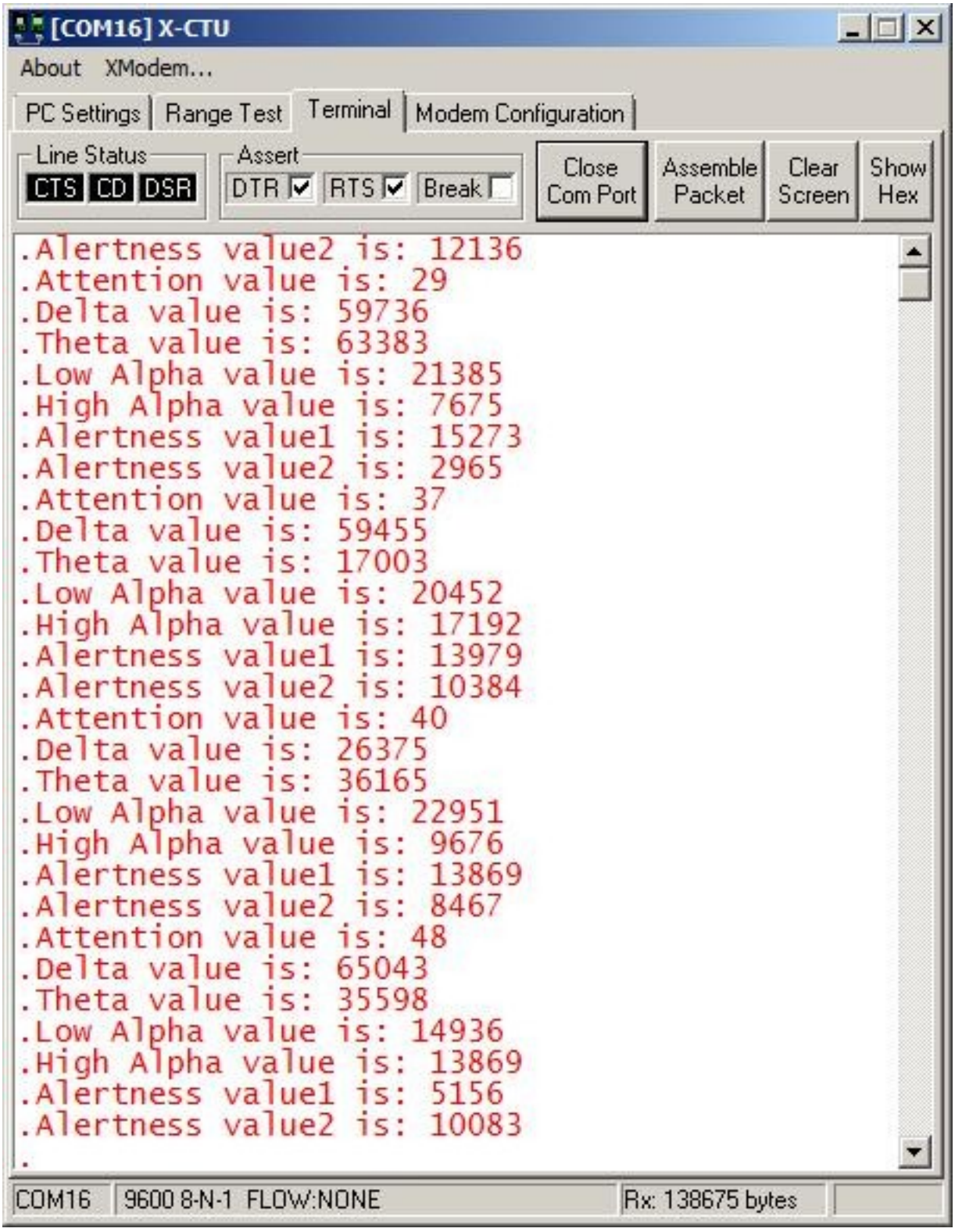
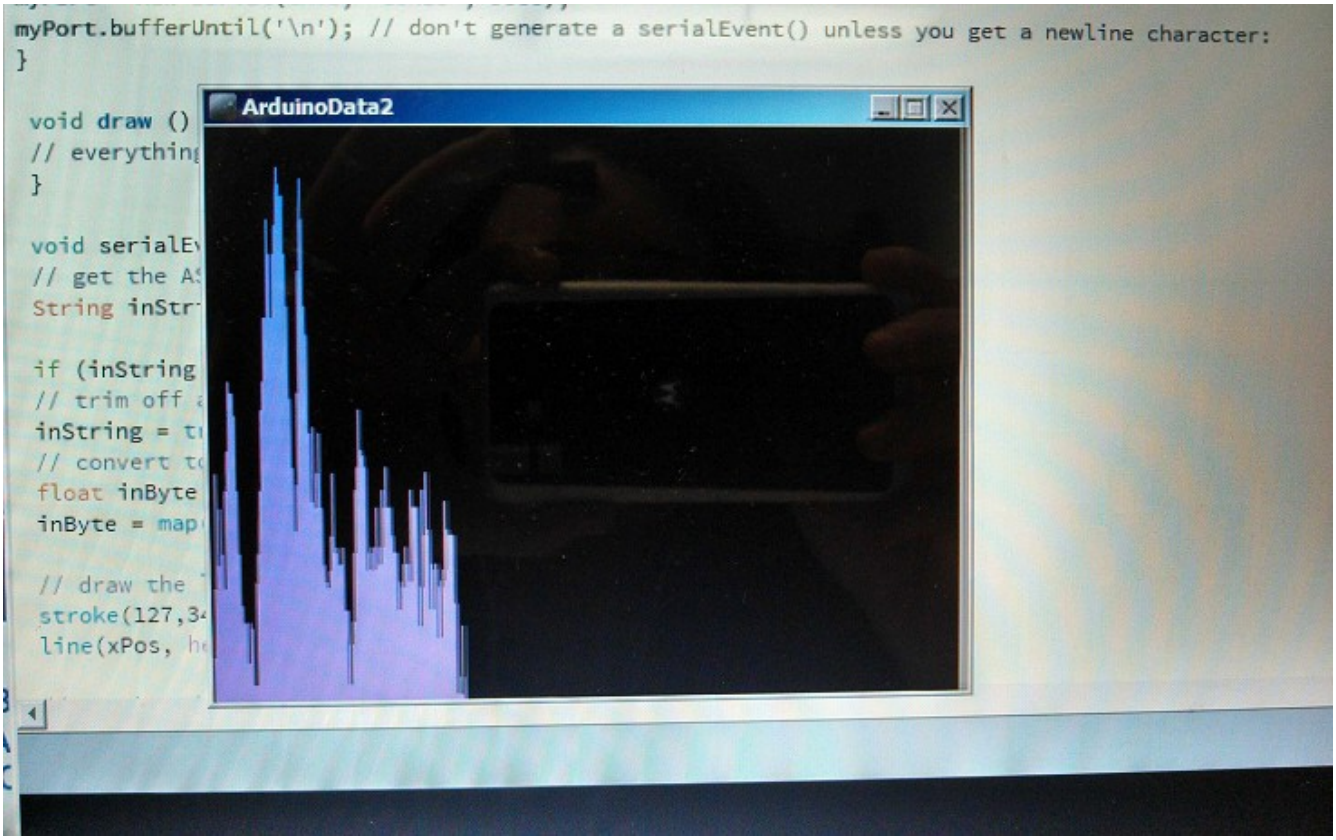


Image via Wikimedia.org

If you used the Arduino code linked above, you should be able to see these values printed out in X-CTU or the Serial terminal program of your choice:



The Processing code linked above can be used and modified to graph different values from the Arduino:



Conclusion

The best results using the MindWave Mobile came when I looked at the processed Attention value. It was easy to see that when I (or anyone else) counted backwards from 100 to 1, the Attention values went up.

The deep brainwave values that I really wanted to see, such as Gamma, Beta, etc., were all over the place, and it was difficult to see any consistency in those values at all.

In fact, the brainwave values varied the same when the unit was on the head as when off the head!

The Android app that comes with the MindWave Mobile didn't work at all, I tried it on a few phones without success. NeuroSky support thought that it was possible that my Android operating system was not compatible.

Overall, I don't think using the NeuroSky MindWave product yields any data that is very usable. I look forward to trying NeuroSky's next revision, and I am really looking forward to trying OpenBCI!

Pt. 5: Connecting Arduino to Processing

Attribution: Originally published by SparkFun, this highly streamlined adaptation of the following Processing tutorial was reconstructed by Mark Maffei via CC BY SA.

Introduction

As a serious Arduino brainhacker, having a basic understanding of how Arduino interacts with Processing is of great benefit. Here's the core essentials of what you need to know to get started.

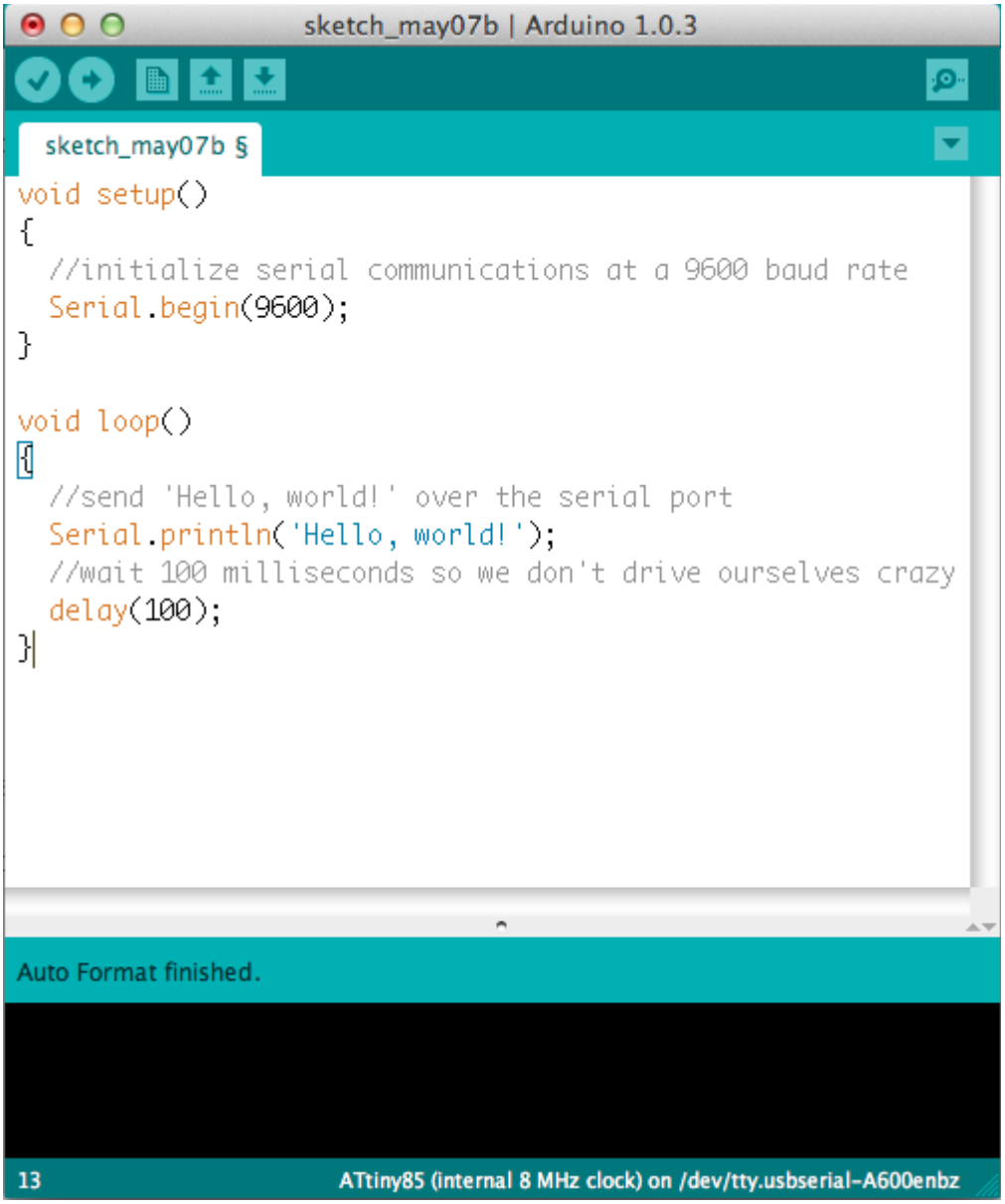
In this tutorial you will learn:

- How to send data from Arduino to Processing over the serial port
- How to receive data from Arduino in Processing
- How to send data from Processing to Arduino
- How to receive data from Processing in Arduino
- How to write a serial 'handshake' between Arduino and Processing to control data flow
- How to make a 'Pong' game that uses analog sensors to control the paddles
- While some basic familiarity with Processing will be useful, it's not strictly necessary.

Starting With The Arduino Side Of Things

Let's start with the basics of how to set up your Arduino sketch to send information over serial. Open up the Arduino software, and copy/paste the code below in.

After doing so, you should see something like this:



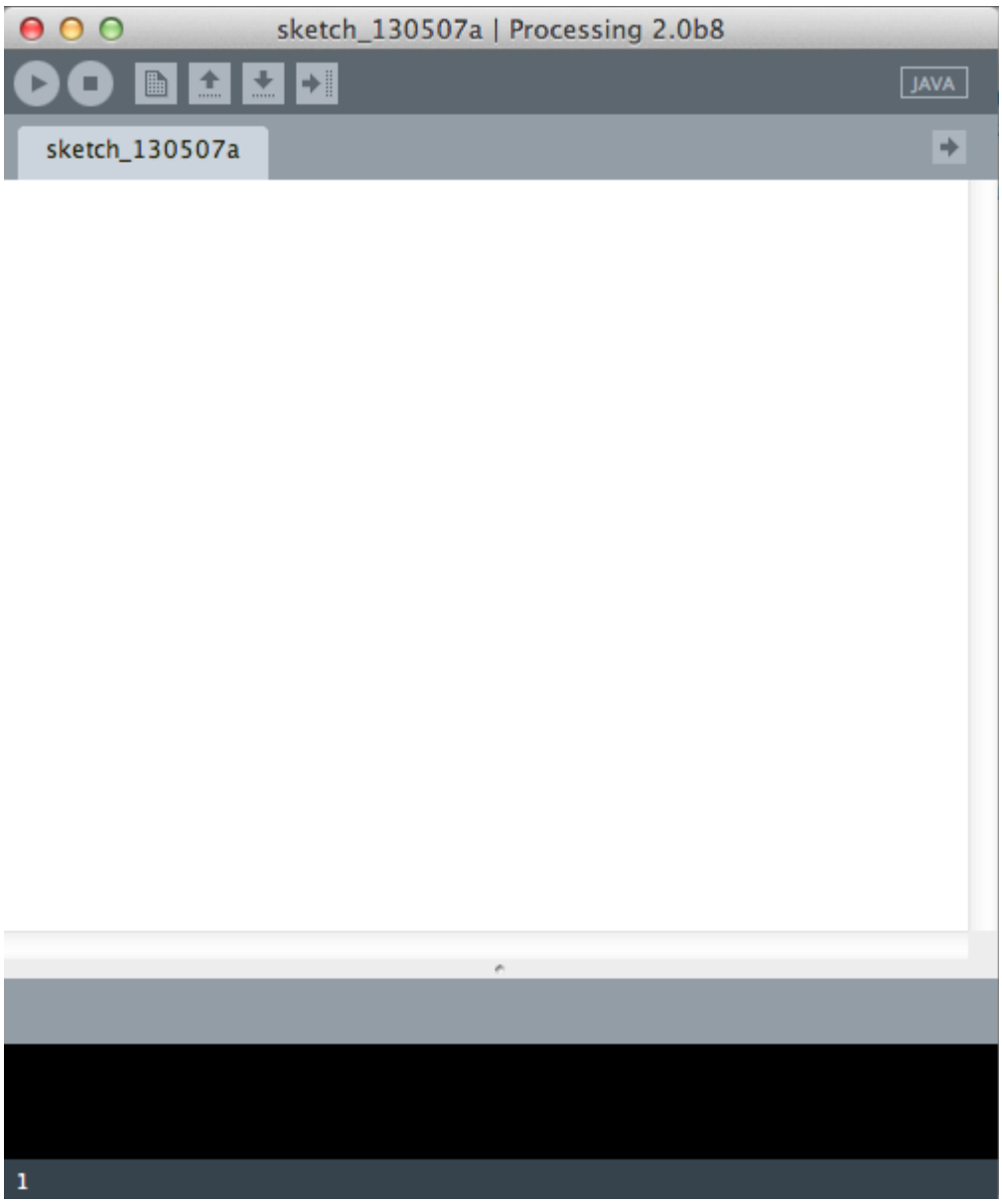
[Get Code Here]

All that's left to do is to plug in your Arduino board, select your board type (under **Tools -> Board Type**) and your Serial port (under **Tools -> Serial Port**) and hit the 'upload' button to load your code onto the Arduino.

Now we're ready to see if we can detect the 'Hello, world!' string we're sending from Processing.

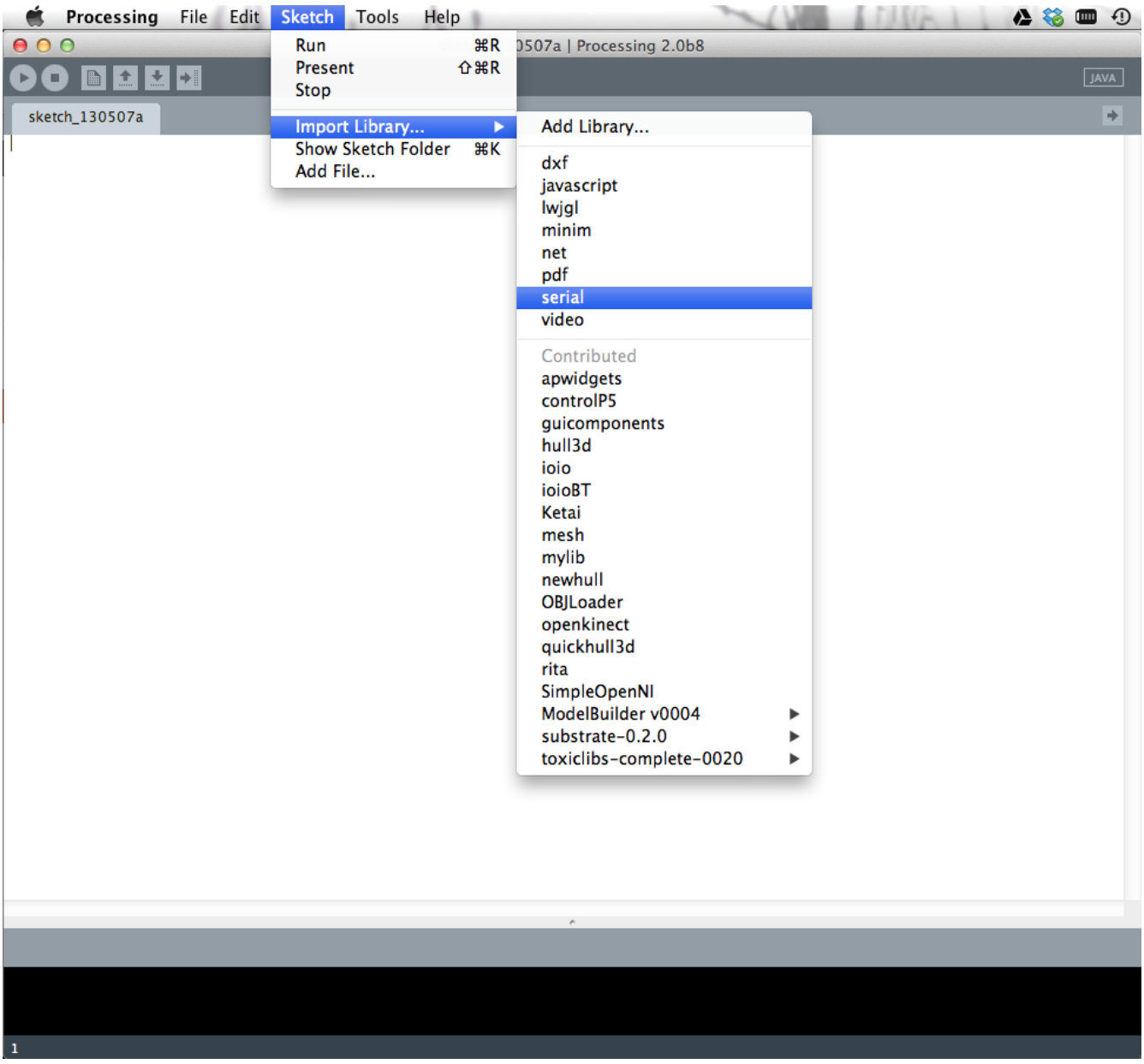
Communicating From Arduino To Processing

If you don't have a version of Processing, make sure you go to Processing.org and download the latest version for your OS. Once Processing is installed, open it up. You should see something like this:



If this looks a lot like the Arduino IDE, it's because the Arduino IDE was actually based in part off of Processing. Ah yes… the beauty of open source!

Ok, so once you have an open sketch, your first step is to import the Serial library. Go to **Sketch->Import Library->Serial**, as shown below:



You should now see a line like import processing.serial.*; at the top of your sketch.

Underneath your import processing.serial.*; at the top of your sketch, copy/paste the code example below (leaving a space between your import statement and the sample code).

A Brief Overview Of What's About To Happen

First, you'll declare some global variables. All this means is that these variables can used anywhere in your sketch. Add these two lines beneath the import statement in your sketch:

```
Serial myPort;  // Create object from Serial class
String val;     // Data received from the serial port
```

Now in order to listen to any serial communication, you'll have to get a Serial object (we call it myPort but you can it whatever you like), which lets you listen in on a serial port on your computer for any incoming data.

You'll also need a variable to receive the actual data coming in. In this case, since you're sending a String (i.e. the sequence of characters 'Hello, World!') from Arduino, you want to receive a String in Processing. Just like Arduino has setup() and loop(), Processing has setup() and draw() (instead of loop).

For your setup() method in Processing to work, you'll find the serial port your Arduino is connected to and set up your Serial object to listen to that port:

```
void setup()
{
  // I know that the first port in the serial list on my mac
  // is Serial.list()[0].
  // On Windows machines, this generally opens COM1.
  // Open whatever port is the one you're using.
  String portName = Serial.list()[0]; //change the 0 to a 1 or 2 etc. to match your port
  myPort = new Serial(this, portName, 9600);
}
```

Remember how you set Serial.begin(9600) in Arduino? Well, like the old saying "What you do to one side, you must also do to the other" definitely applies.

So you need to set 9600 as that last argument in your Serial object in Processing as well, allowing Arduino and Processing to communicate at the same rate.

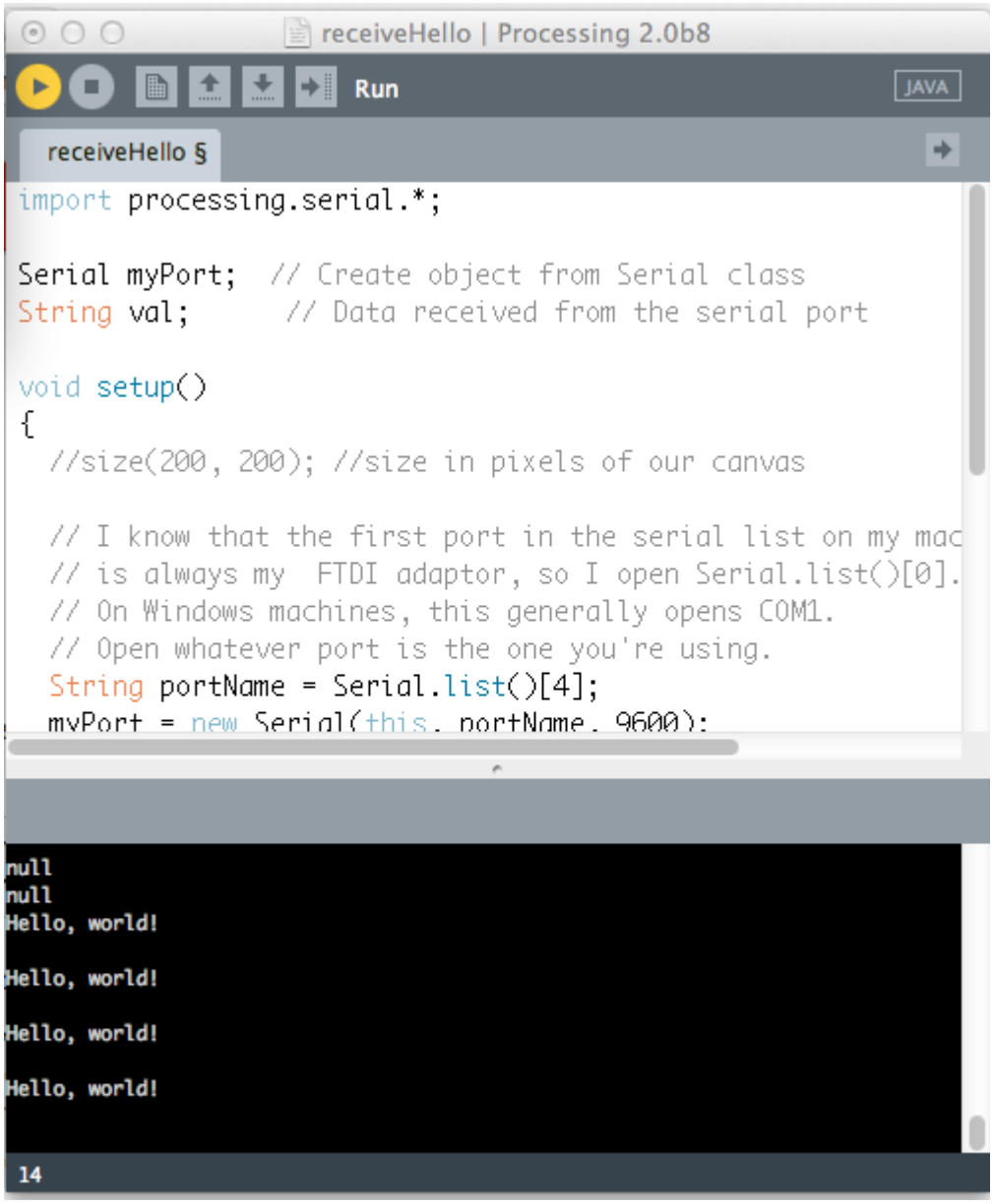
In your draw() loop, you're going to listen in on your Serial port and if you we get something, it'll go in your valvariable and print it to the console (that black area at the bottom of your Processing sketch)…

```
void draw()
{
  if ( myPort.available() > 0)
  {  // If data is available,
  val = myPort.readStringUntil('\n');         // read it and store it in val
  }
println(val); //print it out in the console
}
```

So now that you understand what's going on…

[Get Code Here]

If you hit the ‘run’ button (and your Arduino is plugged in with the code in the previous example loaded up), you should see a little window pop-up, and after a sec you should see `Hello, World!‘ appear in the Processing console. Over and over. Like this:


receiveHello | Processing 2.0b8
Run
JAVA
receiveHello §
import processing.serial.*;
Serial myPort; // Create object from Serial class
String val;   // Data received from the serial port
void setup()
{
//size(200, 200); //size in pixels of our canvas
// I know that the first port in the serial list on my mac
// is always my FTDI adaptor, so I open Serial.list()[0].
// On Windows machines, this generally opens COM1.
// Open whatever port is the one you're using.
String portName = Serial.list()[4];
null
null
Hello, world!
Hello, world!
Hello, world!
Hello, world!
14

Assuming everything went according to plan… you’ve now conquered how to send data from Arduino to Processing. Your next step is figure out how go the opposite way - sending data from Processing to Arduino.

Communicating From Processing To Arduino

The Processing side of things starts out much like your last sketch: you import the Serial library and declare a global Serial object variable for your port up top. Likewise, in your setup() method you find your port and initialize Serial communication on that port with your Serial variable at 9600 baud.

You'll also use the size() command, to give yourself a little window to click in; which will trigger your sketch to send something over the Serial port to Arduino:

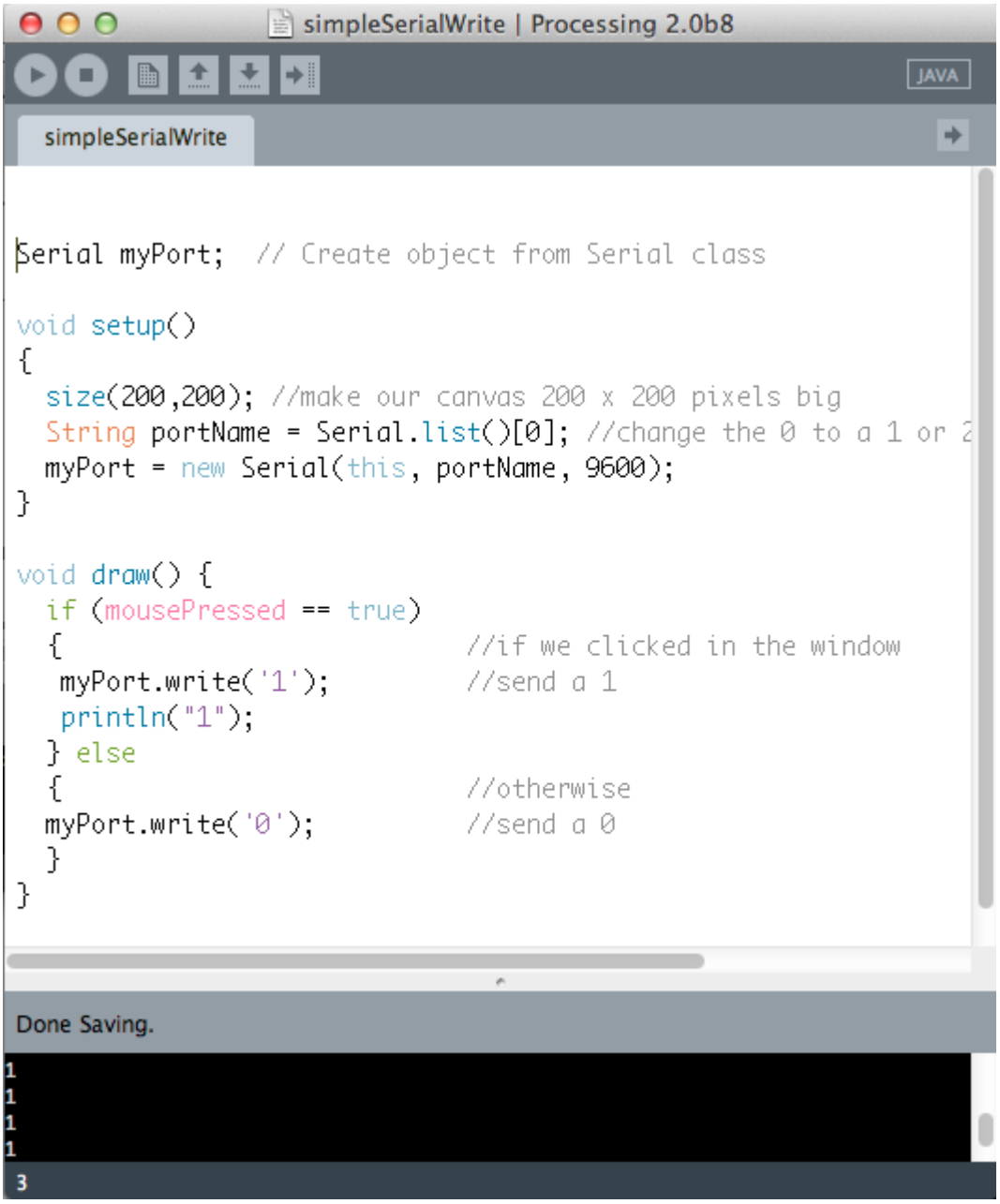
```
import processing.serial.*;

Serial myPort;  // Create object from Serial class

void setup()
{
  size(200,200); //make our canvas 200 x 200 pixels big
  String portName = Serial.list()[0]; //change the 0 to a 1 or 2 etc. to match your port
  myPort = new Serial(this, portName, 9600);
}
```

In your draw() loop, you send whatever you want over the serial port by using the write method from the Processing Serial library. For this sketch, you'll send a '1' whenever you click your mouse in the Processing window. You'll also print it out on the console, just to see that you're actually sending something. If you aren't clicking you'll send a '0' instead:

```
void draw() {
  if (mousePressed == true)
  {                       //if we clicked in the window
   myPort.write('1');         //send a 1
   println("1");
  } else
  {                       //otherwise
  myPort.write('0');          //send a 0
  }
}
```

[Get Code Here]

This is what your code should look like at this point:



If you run this code, you should see a bunch of 1's appear in the console area whenever you click your mouse in the window. But how do you look for these 1's from Arduino… and more importantly - what can you do with them?!

Glad you asked!

Coming Into Arduino

Your Arduino is going to look for those 1's coming in from Processing, and when it sees one, it'll turn on an LED on pin 13 (assuming that your Arduino has pin 13 designated as the on-board LED). This way, you won't need an external LED and resistor to see this work.

At the top of our Arduino sketch, we need two global variables - one for holding the data coming from Processing, and another to tell Arduino which pin our LED is hooked up to:

```
char val; // Data received from the serial port
int ledPin = 13; // Set the pin to digital I/O 13
```

Next, in your setup() method, you'll set the LED pin to an output (since you're powering an LED), and you'll once again start Serial communication at 9600 baud.

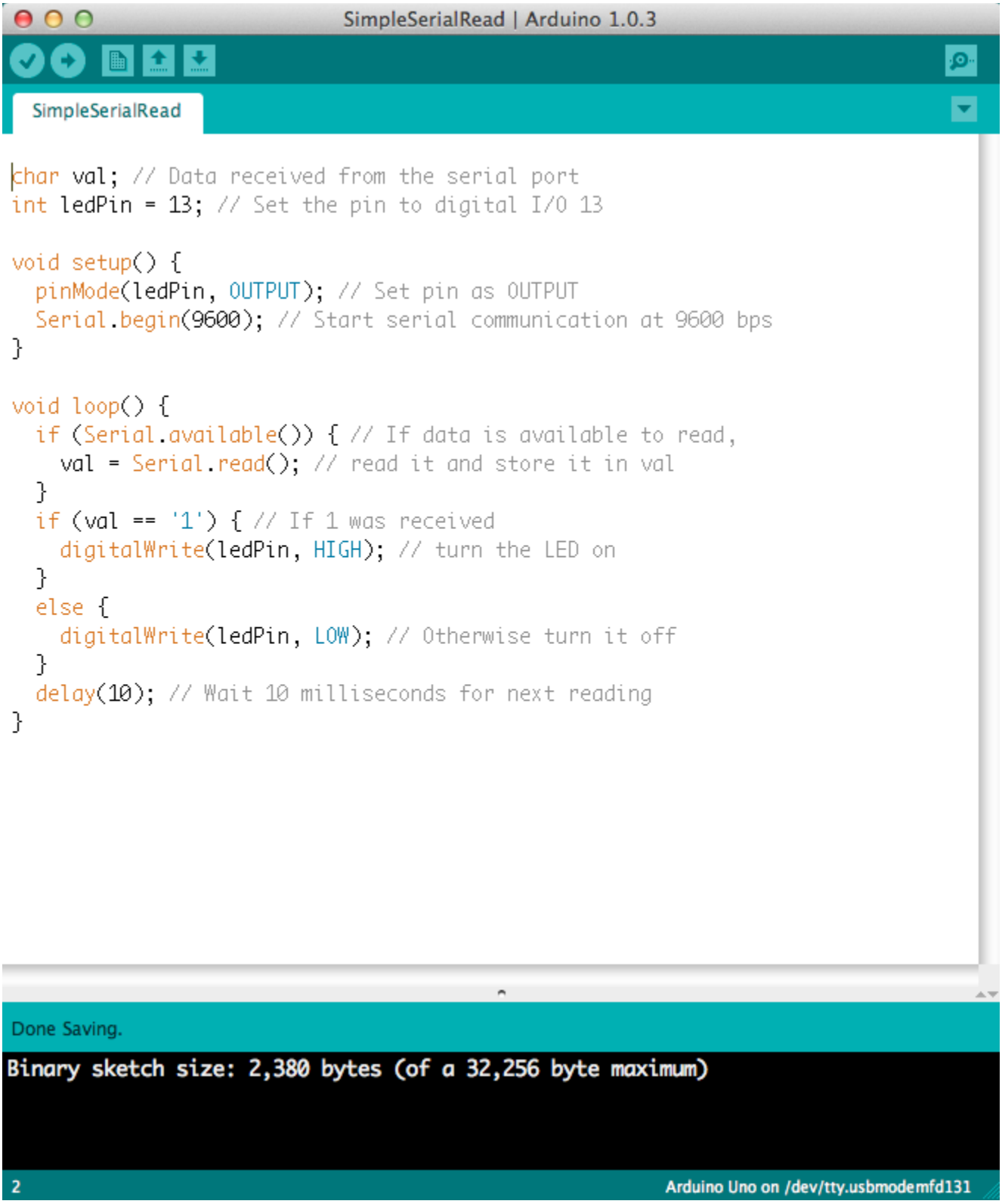
```
 void setup() {
   pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
   Serial.begin(9600); // Start serial communication at 9600 bps
 }
```

Finally, in the loop() method, Arduino looks at the incoming serial data. If a '1', the LED gets set to HIGH (on). If a '0' - the LED gets set to off. At the end of the loop, there's a small delay to help Arduino keep up with the serial stream:

```
 void loop() {
   if (Serial.available())
   { // If data is available to read,
     val = Serial.read(); // read it and store it in val
   }
   if (val == '1')
   { // If 1 was received
     digitalWrite(ledPin, HIGH); // turn the LED on
   } else {
     digitalWrite(ledPin, LOW); // otherwise turn it off
   }
   delay(10); // Wait 10 milliseconds for next reading
}
```

[Get Code Here]

This is what your code should look like, all said and done:


SimpleSerialRead | Arduino 1.0.3
SimpleSerialRead
char val; // Data received from the serial port
int ledPin = 13; // Set the pin to digital I/O 13
void setup() {
pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
Serial.begin(9600); // Start serial communication at 9600 bps
}
void loop() {
if (Serial.available()) { // If data is available to read,
val = Serial.read(); // read it and store it in val
}
if (val == '1') { // If 1 was received
digitalWrite(ledPin, HIGH); // turn the LED on
}
else {
digitalWrite(ledPin, LOW); // Otherwise turn it off
}
delay(10); // Wait 10 milliseconds for next reading
}
Done Saving.
Binary sketch size: 2,380 bytes (of a 32,256 byte maximum)
2
Arduino Uno on /dev/tty.usbmodemfd131

If you load up this code onto your Arduino, and run the Processing sketch from the previous example, and everything goes smoothly… Voila! You should be able to turn on an LED attached to pin 13 of your Arduino, simply by clicking within the Processing canvas.

Shaking Hands

So far we've shown that Arduino and Processing can communicate via serial when one is talking and the other is listening. However, what about making a link that allows data to flow **both** ways; so that Arduino and Processing are *both sending and receiving data*?

It's called a serial 'handshake'; since both sides have to agree when to send/receive data.

From The Arduino Side

We'll now combine our two previous examples in such a way that Processing can both receive 'Hello, world!' from Arduino AND send a 1 back to Arduino to toggle an LED. Of course, this also means that Arduino has to be able to send 'Hello, world!' while listening for a 1 from Processing.

Let's start with the Arduino side of things. In order for this to run smoothly, both sides have to know what to listen for and what the other side is expecting to hear. We also want to minimize traffic over the serial port so we get more timely responses.

Just like in our Serial read example, we need a variable for our incoming data and a variable for the LED pin we want to light up:

```
char val; // Data received from the serial port
int ledPin = 13; // Set the pin to digital I/O 13
boolean ledState = LOW; //to toggle our LED
```

Since we're trying to be efficient, we're going to change our code so that we only listen for 1's, and each time we hear a '1' we toggle the LED on or off. To do this we added a boolean (true or false) variable for the HIGH or LOW state of our LED. This means we don't have to constantly send a 1 or 0 from Processing, which frees up our serial port quite a bit.

Our `setup()` method looks mostly the same, with the addition of an `establishContact()` function which we'll get to later - for now just type it in.

```
void setup()
{
  pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
  //initialize serial communications at a 9600 baud rate
  Serial.begin(9600);
  establishContact();  // send a byte to establish contact until receiver responds
}
```

In our loop function, we've just combined and slimmed down the code from our two earlier sketches. Most importantly, we've changed our LED code to toggle based on our new boolean value.

The '!' means every time we see a one, we set the boolean to the opposite of what it was before (so LOW becomes HIGH or vice-versa). We also put our 'Hello, world!' in an *else statement*, so that we're only sending it when we haven't seen a '1' come in.

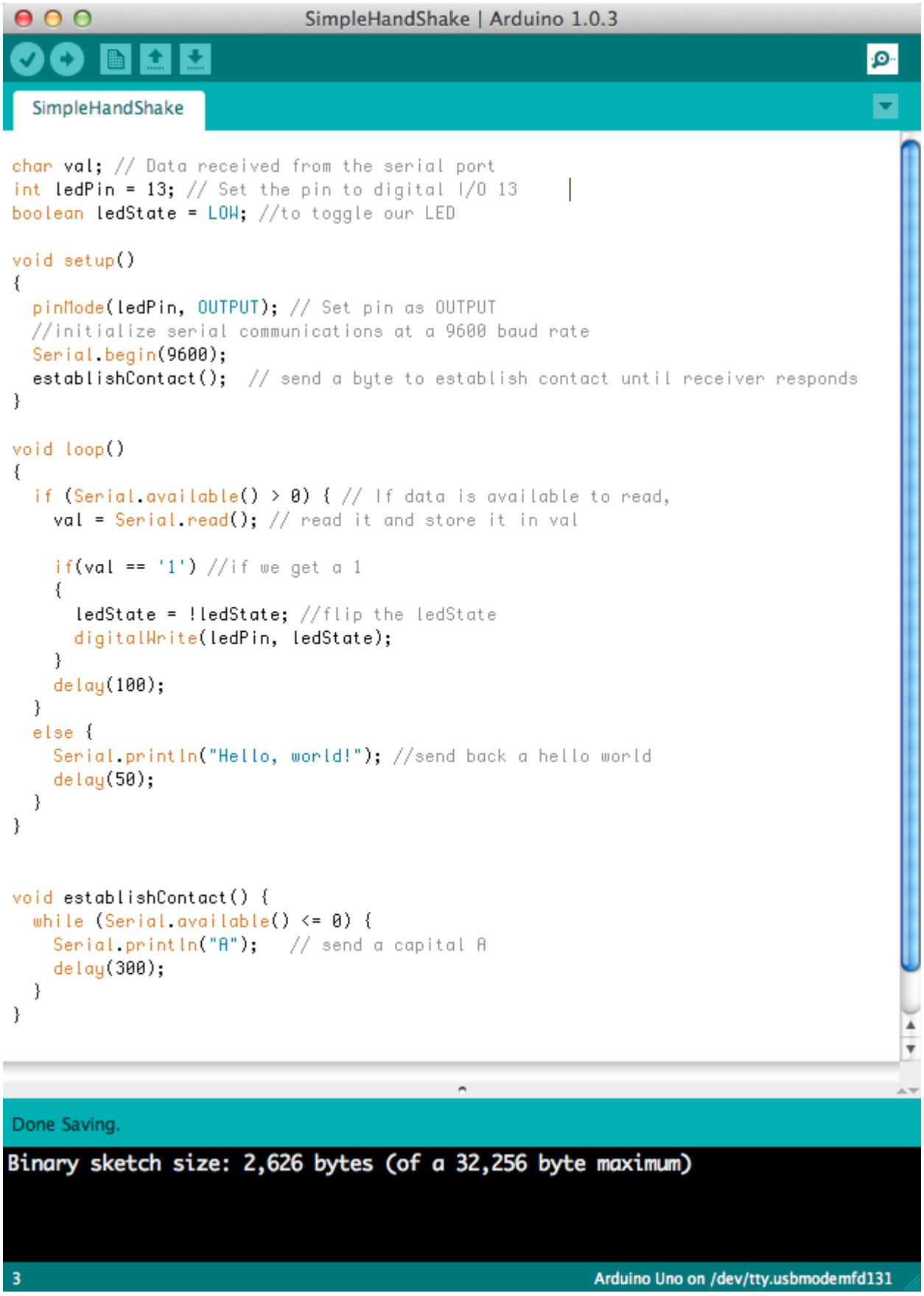
```
void loop()
{
  if (Serial.available() > 0) { // If data is available to read,
    val = Serial.read(); // read it and store it in val

    if(val == '1') //if we get a 1
    {
      ledState = !ledState; //flip the ledState
      digitalWrite(ledPin, ledState);
    }
    delay(100);
  }
    else {
    Serial.println("Hello, world!"); //send back a hello world
    delay(50);
    }
}
```

Now we get to that establishContact() function we put in our setup() method. This function just sends out a string (the same one we'll need to look for in Processing) to see if it hears anything back; indicating that Processing is ready to receive data. It's like saying 'Marco' over and over until you hear a 'Polo' back from somewhere.

```
void establishContact() {
  while (Serial.available() <= 0) {
  Serial.println("A");   // send a capital A
  delay(300);
  }
}
```

[Get Code Here]

Altogether, your Arduino code should look like this:


SimpleHandShake | Arduino 1.0.3
SimpleHandShake
char val; // Data received from the serial port
int ledPin = 13; // Set the pin to digital I/O 13
boolean ledState = LOW; //to toggle our LED
void setup()
{
pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
//initialize serial communications at a 9600 baud rate
Serial.begin(9600);
establishContact(); // send a byte to establish contact until receiver responds
}
void loop()
{
if (Serial.available() > 0) { // If data is available to read,
val = Serial.read(); // read it and store it in val
if(val == '1') //if we get a 1
{
ledState = !ledState; //flip the ledState
digitalWrite(ledPin, ledState);
}
delay(100);
}
else {
Serial.println("Hello, world!"); //send back a hello world
delay(50);
}
}
void establishContact() {
while (Serial.available() <= 0) {
Serial.println("A"); // send a capital A
delay(300);
}
}
Done Saving.
Binary sketch size: 2,626 bytes (of a 32,256 byte maximum)
3
Arduino Uno on /dev/tty.usbmodemfd131

That's it for the Arduino side, now on to Processing!

From The Processing Side

For the Processing side of things, we've got to make a few changes. We're going to make use of the serialEvent() method, which gets called every time we see a specific character in the serial buffer; which acts as our delimiter - basically it tells Processing that we're done with a specific 'chunk' of data - in our case, one 'Hello, world!'.

The beginning of our sketch is the same except for a new firstContact boolean, which let's us know when we've made a connection to Arduino:

```
import processing.serial.*; //import the Serial library
 Serial myPort;  //the Serial port object
 String val;
// since we're doing serial handshaking,
// we need to check if we've heard from the microcontroller
boolean firstContact = false;
```

Our setup() function is the same as it was for our serial write program... *except* we added the myPort.bufferUntil('\n'); line. This let's us store the incoming data into a buffer, until we see a specific character we're looking for.

In this case, it's a carriage return (\n) because we sent a Serial.print *in* from Arduino. The 'ln' at the end means the String is terminated with a carriage return, so we know that'll be the last thing we see:

```
void setup() {
  size(200, 200); //make our canvas 200 x 200 pixels big
  //  initialize your serial port and set the baud rate to 9600
  myPort = new Serial(this, Serial.list()[4], 9600);
  myPort.bufferUntil('\n');
}
```

Because we're continuously sending data, our serialEvent() method now acts as our new draw() loop, so we can leave it empty:

```
void draw() {
  //we can leave the draw method empty,
  //because all our programming happens in the serialEvent (see below)
}
```

Now for the big one: serialEvent(). Each time we see a carriage return, this method gets called; hence we need to do a few things each time to keep things running smoothly:

- Read the incoming data and see if there's actually anything in it (i.e. it's not empty or 'null')
- Trim whitespace and other unimportant stuff
- If it's our first time hearing the right thing, change our firstContact boolean and let Arduino know we're ready for more data
- If it's *not* our first run, print the data to the console and send back any valid mouse clicks (as 1's) we got in our window
- Finally, tell Arduino we're ready for more data

That's a lot of steps, but luckily for us Processing has functions that make most of these tasks pretty easy. Let's take a look at how it all breaks down:

```
void serialEvent( Serial myPort) {
//put the incoming data into a String -
//the '\n' is our end delimiter indicating the end of a complete packet
val = myPort.readStringUntil('\n');
//make sure our data isn't empty before continuing
if (val != null) {
  //trim whitespace and formatting characters (like carriage return)
  val = trim(val);
  println(val);

  //look for our 'A' string to start the handshake
  //if it's there, clear the buffer, and send a request for data
  if (firstContact == false) {
    if (val.equals("A")) {
      myPort.clear();
      firstContact = true;
      myPort.write("A");
      println("contact");
    }
  }
  else { //if we've already established contact, keep getting and parsing data
    println(val);

    if (mousePressed == true)
    {                       //if we clicked in the window
      myPort.write('1');        //send a 1
      println("1");
    }

    // when you've parsed the data you have, ask for more:
```

```
    myPort.write("A");
    }
  }
}
```

Whew… that's a lot to chew on. But if you read carefully line by line (especially the comments), it'll start to make sense. If you've got your Arduino code finished and loaded onto your board, try running this sketch.

You should see 'Hello, world!' coming in on the console, and when you click in the Processing window, you should see the LED on pin 13 turn on and off. Success! You are now a serial handshake expert.

[Get Code Here]

Tips and Tricks

In developing your own projects with Arduino and Processing, there are a few 'gotchas' that are helpful to keep in mind in case you get stuck.

- Make sure your baud rates match.
- Make sure you're reading off the right port in Processing - there's a `Serial.list()` command that will show you all the available ports you can connect to.
- If you're using the `serialEvent()` method, make sure to include the `port.bufferUntil()` function in your `setup()` method.
- Also, make sure that whatever character you're buffering.
- If you want to send over a number of sensor values, it's a good idea to count how many bytes you're expecting so you know how to properly parse out the sensor data (the example shown below that comes with Arduino) gives a great example of this).

Going Further

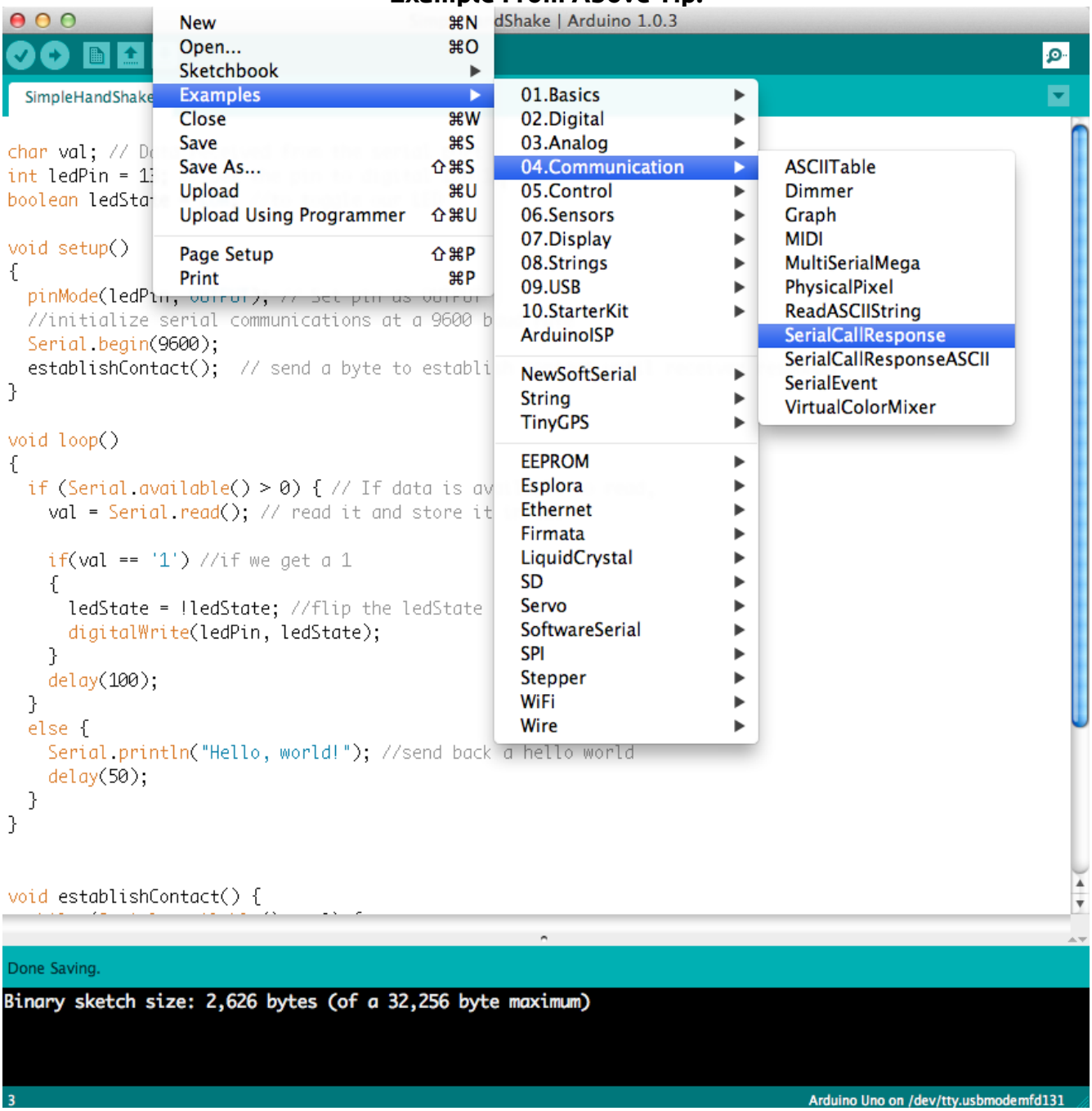
Now that you know how to send data from Arduino to Processing and back again (even simultaneously!)… you're ready for some seriously cool projects.

By hooking together Arduino and Processing, you can do things like visualize sensor data in real-time, or make a glove with flex sensors in the fingers that makes penguins appear on the screen, or a command console from Processing that controls a giant array of LEDs.

Here are a few Processing links to increase your knowledge going forward:

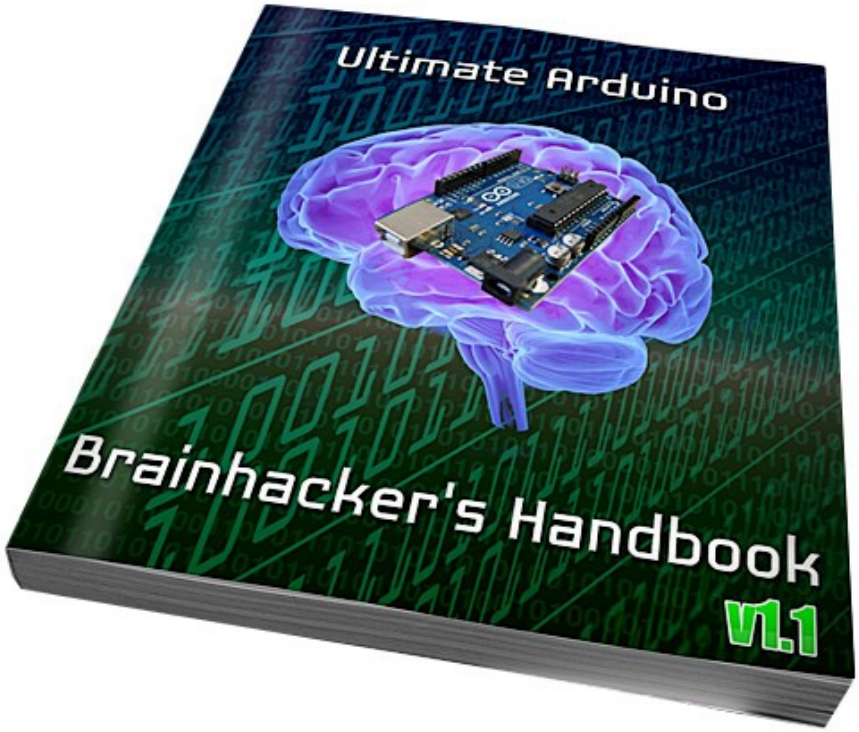
- [Derek Runberg's Processing Curriculum]
- [Arduino & Processing ATLAS Curriculum]
- [Processing the Danger Shield]
- [Arduino, Processing, & MaxMSP]

Example From Above Tip:



Stay Tuned For

V1.1



ArduinoRocksMe.WordPress.com

Contents

Published By Mark Maffei